Cooperative Target Following and Collision Avoidance for Multiple Unmanned Aerial Vehicles

by

Johan B. Ubbink

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University.

Supervisor: Dr JAA Engelbrecht

December 2020

Acknowledgements

I want to thank the following people most sincerely:

- My supervisor, Dr Japie Engelbrecht, for his enthusiasm, guidance, wisdom and interest throughout the project.
- Anne Erikson, for assisting in the language editing.
- The Harry Crossley foundation, for their financial support.
- Everyone at the Electronic Systems Laboratory (ESL), for the good times at the office, as well as the support and friendship.
- My family - Onno, Marike, Anja, Carla and Paul for all their support, love and help along the way.
- My wife, Marli, for all her love, patience and support.

Plagiarism Declaration

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

JB Ubbink	
Handtekening / *Signature*	Studentenommer / *Student number*
JB Ubbink	9 October 2020
Voorletters en van / *Initials and surname*	Datum / *Date*

Cooperative Target Following and Collision Avoidance for Multiple Unmanned Aerial Vehicles

Abstract

In the field of robotics, there is an increasing number of applications for multiple Unmanned Aerial Vehicles (UAVs) to operate autonomously in a complex environment. This study contributes to the research field by presenting a trajectory planner for multiple rotary-wing UAVs to follow a moving ground vehicle cooperatively and collision-free through a three-dimensional domain with arbitrary static obstacles.

An optimal control formulation is used to represent the target-following problem. The optimal control objective function is designed to minimise the position error in target following as well as the acceleration of the UAVs. Constraints are imposed to ensure that the trajectories are dynamically feasible and to avoid collisions between UAVs or between UAVs and obstacles.

The optimal control problem is transcribed to a Nonlinear Program (NLP) and solved with the aid of a general optimisation solver. The optimisation solver starts with suitable initial estimates of the trajectories, which it iteratively improves until the optimal trajectories are obtained. These initial estimates are calculated with grid searches (based on the A* algorithm) that independently plan trajectories for all the UAVs. Each grid search neglects the other UAVs but does consider static obstacles in the environment. This decoupling (neglecting other UAVs) allows for initial estimates to be computed in parallel. The solver also requires that the objective function and constraint functions are smooth and differentiable. The static obstacles in the optimisation problem are represented by Euclidean Signed Distance Fields (ESDFs), which gives the distance to the nearest obstacle.

The trajectory planner reacts to changes in the predicted target trajectory by using a replanning strategy similar to a Model Predictive Control (MPC) strategy. The replanning strategy continuously (at a fixed rate) plans for a receding horizon into the future. The trajectories are planned to ensure safe execution, even if an iteration of the replanning strategy fails. In-depth analysis of specific design parameters (planning resolution, planning horizon length and replanning rate) was performed, both from a theoretical perspective as well as simulation experiments. The analysis shows that some of the parameters are conflicting, and it is essential to balance the parameters to obtain a viable real-time implementation.

The trajectory planner was combined with other components within the Robot Operating System (ROS), to form a target following system. Special care has been taken to ensure that the implementation could serve as a research platform for future projects in which multi-agent robotics systems can be developed and tested.

The performance of the proposed trajectory planner is tested through simulation. First, examples are presented to illustrate the trajectory planner and target-following system. The trajectory planner is also tested in a large number of randomly generated scenarios, varying in complexity. The performance is analysed in terms of success rate, target following ability and planning time. The simulation results show that the UAVs can successfully follow a moving ground target while avoiding collisions with one another and with static obstacles for a large variety of targets and environments.

Opsomming

In die veld van robotika is daar 'n toenemende aantal toepassings vir verskeie onbemande vliegtuie om saam te werk op 'n gegewe opdrag. Hierdie studie dra by tot die navorsingsveld deur 'n trajekbeplanner te ontwerp vir verskeie rotor-vlerk vliegtuie om saam 'n teiken op die grond te agtervolg, terwyl hulle botsings met mekaar en die omgewing vermy.

Die teikenvolging probleem word voorgestel as 'n optimale beheerprobleem. Die koste funksie is ontwerp om die volgingsfout sowel as onnodige versnelling deur die vliegtuie te minimeer. Die optimeringsprobleem word beperk om te verseker dat die trajekte uitvoerbaar is, sowel as om botsings tussen vliegtuie en botsings met die omgewing te vermy.

Met die gebruik van trajekoptimeringstegnieke word die optimale beheerprobleem na 'n nie-lineêre program (NLP) oorgeskryf en met behulp van 'n algemene optimerings oplosser opgelos. Die optimerings oplosser benodig 'n geskikte aanvanklike afskatting van die trajek, en die afgeleide van al die funksies moet bereken kan word. Die projek ondersoek die gebruik van 'n soekalgoritme om die aanvanklike skatting te gee. Die soekstrategie neem nie die ander vliegtuie in die omgewing in ag nie, maar vermy hindernisse in die omgewing. Deurdat die vliegtuie nie mekaar in ag neem nie, kan aanvanklike afskattings in parallel bereken word. Die statiese omgewing word voorgestel deur 'n veld wat die afstand tot die naaste hindernes aandui.

Die projek pas die konsep van modelvoorspelling-beheerstrategie (MPC) toe om die trajekte intyds te beplan. Die MPC-strategie beplan voortdurend (teen 'n vaste tempo) vir 'n horison in die toekoms in. Die trajekte word so beplan dat dit veilige uitvoering verseker, selfs as 'n iterasie van die trajekbeplanner faal. 'n Analiese is gedoen om die inpak van die beplanningsresolusie, beplanningshorison en herbeplanningstempo te meet. Die resultate wys dat die veranderlikes in stryd is met mekaar, en versigtig gekies moet word om die trajekte intyds te beplan.

Die beplanner is geïmplementeer in die Robotic Operating System (ROS) en getoets in samewerking met 'n trajek uitvoerder in 'n Gazebo-simulasie. Afgesien van die spesifieke toepassings wat uiteengesit en getoets is, is daar veral gesorg dat die implementering kan dien as 'n navorsingsplatform vir toekomstige projekte waarin robotiese stelsels ontwikkel en getoets kan word.

Verskeie gevallestudies word aangebied om spesifieke kenmerke van die trajekbeplanner uit te lig en om die prestasie te evalueer. Die resultate toon dat die beplanner in staat is om botsingsvrye trajekte vir 'n verskeidenheid teikens en omgewings te genereer.

Contents

Declaration ii

Abstract iii

Opsomming iv

List Of Figures ix

List Of Tables xii

Nomenclature xiii

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Target-following system . . . 3
  - 1.2.1 Target . . . 3
  - 1.2.2 Environment . . . 4
  - 1.2.3 Trajectory planner . . . 4
  - 1.2.4 Trajectory execution . . . 5
  - 1.2.5 Communication . . . 6
- 1.3 Research aim and methodology . . . 6
- 1.4 State of current research . . . 6
- 1.5 Primary contributions . . . 7
- 1.6 Research scope . . . 8
- 1.7 Thesis outline . . . 9

2 Literature Review 10
- 2.1 Target following . . . 10
  - 2.1.1 Single UAV target following . . . 10
  - 2.1.2 Multiple fixed-wing UAVs . . . 11
  - 2.1.3 Multiple rotary-wing UAVs . . . 11
  - 2.1.4 Autonomous UAV navigation . . . 11
  - 2.1.5 Formation control . . . 12
  - 2.1.6 Conclusion . . . 12
- 2.2 Overview of trajectory-planning techniques . . . 13
  - 2.2.1 Concepts and definitions . . . 13
  - 2.2.2 Grid-based search . . . 13
  - 2.2.3 Sampling-based methods . . . 14
  - 2.2.4 Artificial potential field methods . . . 15

2.2.5 Optimal control . . . 15
2.3 Multi-agent planning techniques . . . 18
2.3.1 Global approach . . . 18
2.3.2 Decoupled approach . . . 19
2.3.3 Hybrid approaches . . . 19
2.4 Related concepts . . . 20
2.4.1 Trajectory tracking . . . 20
2.4.2 Target state estimation and prediction . . . 21
2.4.3 Mapping . . . 22
2.5 Summary . . . 23

3 Technique Selection **24**
3.1 Technique requirements . . . 24
3.2 Planning technique . . . 25
3.3 Transcription approach . . . 26
3.3.1 Indirect methods vs direct methods . . . 26
3.3.2 Shooting methods vs collocation methods . . . 26
3.4 Incorporating obstacles . . . 28
3.4.1 Map representation . . . 29
3.4.2 Obstacle constraints in trajectory optimisation . . . 30
3.5 Enforcing dynamics . . . 31
3.6 Global planner vs decoupled planner . . . 31
3.7 Replanning . . . 32
3.8 NLP Libraries . . . 33
3.9 Mathematics of solving NLPs . . . 33
3.9.1 Newton's method for optimisation . . . 34
3.9.2 Constrained optimisation . . . 36
3.10 Summary . . . 37

4 Trajectory Planner Design **39**
4.1 Overview . . . 39
4.2 Trajectory planner architecture . . . 40
4.3 Modelling . . . 41
4.3.1 Ground vehicle (Target) . . . 41
4.3.2 Rotary-wing UAVs . . . 41
4.3.3 Obstacle representation . . . 42
4.4 Optimisation problem formulation . . . 42
4.4.1 Decision variables . . . 43
4.4.2 Objective function . . . 43
4.4.3 Constraints . . . 44
4.5 NLP Solver . . . 46
4.6 Initial estimate . . . 46
4.6.1 Formulation . . . 48
4.6.2 A* Algorithm . . . 49
4.7 Interpolation . . . 50
4.8 Desired yaw angle . . . 51
4.9 Online target following . . . 51
4.9.1 Replanning strategy . . . 52
4.9.2 Recursive feasibility . . . 53

4.9.3 Handling planning errors . . . 54
4.10 Trajectory planning example . . . 54
4.11 Summary . . . 58

5 System Implementation **59**
5.1 Architecture . . . 59
5.2 Trajectory execution . . . 59
5.2.1 Rotary-wing UAV . . . 60
5.2.2 Attitude controller . . . 61
5.2.3 Trajectory tracker . . . 61
5.3 Communication . . . 62
5.3.1 The ROS environment . . . 62
5.3.2 ROS concepts . . . 63
5.4 Simulation environment . . . 63
5.4.1 Environment maps . . . 63
5.4.2 Ground target . . . 65
5.4.3 Rotorwing UAV simulation model . . . 66
5.5 Summary . . . 67

6 Parameter Selection **68**
6.1 Parameter investigation . . . 68
6.1.1 Vehicle velocity and acceleration . . . 68
6.1.2 Planning resolution . . . 69
6.1.3 Planning horizon . . . 70
6.1.4 Replanning rate . . . 71
6.2 Parameter experiments . . . 71
6.2.1 Performance metrics . . . 72
6.2.2 Experiment 1: Planning resolution . . . 72
6.2.3 Experiment 2: Planning horizon . . . 74
6.2.4 Experiment 3: Replanning rate . . . 75
6.3 Summary . . . 76

7 Simulation Results **78**
7.1 System requirements . . . 78
7.2 Requirements testing approach . . . 78
7.3 Target following examples . . . 79
7.3.1 Example: No obstacles (desert environment) . . . 79
7.3.2 Example: With obstacles (forest environment) . . . 80
7.3.3 Example: Target following system . . . 82
7.4 Trajectory planning testing . . . 85
7.4.1 Simulation test setup . . . 86
7.4.2 Simulation test results . . . 86
7.4.3 Success rate . . . 87
7.5 Results analysis . . . 90
7.5.1 Failure cases . . . 91
7.6 Summary . . . 92

8 Conclusion **94**
8.1 Summary of work . . . 94
8.2 Key findings . . . 95
8.3 Future research opportunities . . . 96
8.3.1 Trajectory planning . . . 96
8.3.2 Target following system . . . 97
8.4 Reflection . . . 97

Bibliography **99**

A Constraint Optimisation **106**
A.1 Constraints . . . 106
A.1.1 Equality constraints . . . 106
A.1.2 Inequality constraints . . . 107
A.2 Interior-point method . . . 107
A.3 Globalisation strategies . . . 108
A.3.1 Merit function . . . 108
A.3.2 Line search . . . 109
A.3.3 Filters . . . 109
A.4 Gradient calculation . . . 110

B Rotary-wing UAV dynamics **112**
B.1 Mathematical model . . . 112

List of Figures

1.1 Flycon motion-capture system 1
1.2 Multiple UAVs tasked with autonomously following a moving target . . 2
1.3 Architecture of a target-following system. 3
1.4 Definition of distance d and angle α 4
1.5 Target position feedback control 4
1.6 AscTec Firefly hexarotor 5

2.1 Quadcopter following moving target by dividing the area into grids. . . 14
2.2 Probabilistic Road Map (PRM) for planning trajectories 14
2.3 Rapidly-exploring Random Trees (RRT) for planning trajectories . . . 15
2.4 Artificial potential field method of planning trajectories. 15
2.5 A comparison between open-loop and closed-loop solutions 16
2.6 Global approach to trajectory planning. 18
2.7 Decoupled approach to trajectory planning 19
2.8 Occupancy grid representation of an environment. 22

3.1 Shooting approach for solving a trajectory optimisation problem. . . . 27
3.2 Collocation approach for solving a trajectory optimisation problem. . . 28
3.3 IRIS algorithm for computing obstacle free regions 29
3.4 Euclidean signed distance field of the environment. 30
3.5 Receding horizon control strategy of MPC controller 32
3.6 Objective function with single optimisation variable. 34
3.7 Objective function with 2nd order approximation. 35
3.8 Comparison of objective functions in terms of convergence. 35
3.9 Local minima and global minima of the optimisation problem. 36
3.10 Lagrange multiplier method for constrained optimisation 36

4.1 Summary of trajectory planning process. 39
4.2 Architecture of the proposed trajectory planner. 40
4.3 Coordinate frame of the target. 41
4.4 Coordinate frames of the target and the UAVs in the world frame. . . . 42
4.5 Decision variables for two agents 43
4.6 Definition of distance d and angle α 43
4.7 Constraints between the obstacles and UAVs. 45
4.8 Two strategies for initialising the optimisation algorithm. 46
4.9 Initial estimate going through obstacle. 47
4.10 Initial estimate going around obstacle. 47
4.11 Planning paths sequentially for agents. 47
4.12 Planning paths in parallel over multiple threads. 48

4.13 Strategies for mounting a camera on a UAV 51
4.14 Definition of the desired yaw angle 52
4.15 Replanning strategy to replan trajectories online 52
4.16 Trajectory planner timing diagram 53
4.17 Example scenario to illustrate the operation of the trajectory planner. . 55

5.1 Architecture of a target following system 60
5.2 Architecture of the trajectory execution component 60
5.3 AscTec Firefly hexarotor 61
5.4 Architecture of the attitude controller. 61
5.5 Example of a ROS architecture 63
5.6 Four UAVs following a ground target through a test environment . . . 64
5.7 Randomly generated targets for simulation environment 66
5.8 Simulation framework of the RotorS Simulator 67

6.1 2D Free body diagram of rotary-wing UAV. 69
6.2 Effect of planning resolution on trajectory control. 69
6.3 High-resolution vs low-resolution path around an obstacle point. 70
6.4 Worst case collision constraint violation for low resolution planning. . . 70
6.5 Success rate as a function of planning resolution. 73
6.6 Actual minimum distance to nearest obstacle for each simulation. . . . 73
6.7 Planning time vs planning resolution. 74
6.8 Objective function vs planning resolution. 74
6.9 Success rate vs horizon length. 75
6.10 Planning time vs horizon length. 75
6.11 Target-following error vs horizon length. 75
6.12 Success rate vs the replanning rate. 76
6.13 Planning time vs the replanning rate. 76
6.14 Objective function vs the replanning rate. 76

7.1 Multiple UAVs following a target through the Desert 80
7.2 Position error in following target for scenario with two UAVs 80
7.3 Multiple UAVs following a target through the Forest 81
7.4 Distance to the nearest UAV and obstacle 81
7.5 Position error for two UAVs following target 82
7.6 Trajectories for UAVs following a target 83
7.7 Collision clearance for target following system 84
7.8 Definition of cross-track, along-track and vertical error 84
7.9 Trajectory-tracking error for UAV 2 following the ground vehicle. . . . 85
7.10 Tracking error in executing trajectory. 86
7.11 Breakdown of success rate for trajectory planner 87
7.12 Breakdown of target following error 88
7.13 Total planning time for different numbers of agents. 89
7.14 Correlation between planning time and the number of iterations. 90
7.15 Grid search providing insufficient initial estimate 91
7.16 Target changing direction in front of obstacle 92

A.1 Barrier parameter for each iteration. 108
A.2 Line search filter for strategy 110

A.3 Line-search filter for constraint optimisation. 110
A.4 Expression graph for performing automatic differentiation. 111

B.1 Forces and moments acting on the center of a single rotor 112
B.2 Quadrotor with the body frame B and the global world frame W 113

List of Tables

3.1 Summary of popular NLP solvers 33

4.1 Path at different iterations 55

5.1 Categories of simulation environments 64

6.1 Base parameters for experiments 72
6.2 Parameters for optimisation 77

A.1 Expression graph for calculating derivative 111

Nomenclature

Abbreviations

3D	three-dimensional
AD	Automatic Differentiation
ADMM	Alternating Direction Method of Multipliers
APF	Artificial Potential Field
BVP	Boundary Value Problem
CHOMP	Covariant Hamiltonian Optimisation for Motion Planning
EKF	Extended Kalman Filter
EPL	Eclipse Public License
ESDF	Euclidean Signed Distance Field
FIESTA	Fast Incremental Euclidean diSTAnce
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IPOPT	Interior Point OPTimizer
IRIS	Iterative Regional Inflation by Semidefinite programming
MAV	Micro Aerial Vehicle
MPC	Model Predictive Control
NLP	Non-linear Program
OA	Optimal Anytime
PANOC	Proximal Averaged Newton-type method for Optimal Control
PD	Proportional and Derivative
PID	Proportional, Derivative and Integral
PRM	Probabilistic Road Map
QP	Quadratic Program

ROS	Robotic Operating System
RRG	Rapidly-exploring Random Graph
RRT	Rapidly-exploring Random Trees
SLAM	Simultaneous Localisation And Mapping
SQP	Sequential Quadratic Programming
TSDF	Truncated Signed Distance Field
UAV	Unmanned Aerial Vehicles
WHCA*	Windowed Hierarchical Cooperative A*

Symbols and functions - Trajectory planning

A	Number of agents
$\boldsymbol{f}$	State transition function
f_{MPC}	Replanning rate of the replanning strategy, defined as $f_{MPC} = \frac{1}{t_{MPC}}$
N	Number of collocation grid points
p	Length of prediction horizon
t_s	Planning resolution
t_{MPC}	Replanning interval of the replanning strategy
$\mathbf{u}$	Input to the system
$\mathbf{x}$	State of the system

Symbols and functions - Target modelling

d	UAV distance from the ground target
h	Height of UAV relative to the ground target
$\mathbf{t}$	Position of the ground target
v	Velocity of target along its body axis
α	UAV angle relative to the ground target heading
ψ_{target}	Heading angle of the ground target

Symbols and functions - UAV modelling

a	Acceleration of the UAV
$\mathbf{p}$	Position of the UAV in world axis
$\mathbf{q}$	Orientation of the UAV in world axis

$\mathbf{v}$	Velocity of a UAV in body axis
ϕ	Roll angle of UAV
ψ	Yaw angle of UAV
θ	Pitch angle of UAV
$\boldsymbol{\sigma}$	Differentially flat inputs $[x, y, z, \psi]^\intercal$

Symbols and functions - Optimisation theory

α	Step size of line search
$\boldsymbol{c}$	Equality constraints including slack variables
$\mathcal{F}$	Filter points list of optimisation algorithm
F	Objective function of the optimisation problem
$\boldsymbol{g}$	Inequality constraints of the optimisation problem
$\boldsymbol{g}^L$	Lower bound of the inequality constraints
$\boldsymbol{g}^U$	Upper bound of the inequality constraints
$\boldsymbol{h}$	Equality constraints of the optimisation problem
λ	Lagrange multiplier for constraint optimisation
$\mathcal{L}$	Lagrangian function for constraint optimisation
M	Merit function for determining optimiser progress
$\boldsymbol{p}$	Direction of the newton step
s	Slack variables within the optimisation problem
v	Constrain violation of filter point
$\boldsymbol{x}$	Decision variables of the optimisation problem
$\boldsymbol{x}^L$	Lower bound of the decision variables
$\boldsymbol{x}^U$	Upper bound of the decision variables

Chapter 1

Introduction

1.1 Background

Robotics is an interdisciplinary research area at the interface of engineering and computer science, creating devices that can operate autonomously [1]. One such application is rotary-wing Unmanned Aerial Vehicles (UAVs) following a moving ground target autonomously through a domain with obstacles. This is referred to as autonomous *target following*.

Typical applications for such a system are motion capture, aerial footage for filming, target tracking, and surveillance [2]–[4]. For example, Figure 1.1 illustrates an outdoor scenario of a motion-capture system where cameras are fitted on different UAVs in order to capture the motion of a human gait [5]. To accurately capture this motion, Nägeli *et al.* [5] opted to use a system where a minimum of two UAVs are required. However, their implementation is limited to obstacle-free (collision-free) environments, as illustrated in Figure 1.1.

Figure 1.1: The Flycon motion-capture system developed by Nägeli *et al.* [5] in which cameras on UAVs are used for estimating the motion of a moving subject.

Target following with multiple drones in cluttered environments, while avoiding collisions, is a daunting task. Not only do the UAVs need to follow the target autonomously, but they also need to avoid the obstacles in the 3D domain. Additionally, they also need to avoid collisions with one another. A vital component in robotics for avoiding collisions is *trajectory planning*. Trajectory planning is the task of generating

trajectories that describe where each robot should be at each time step to achieve a goal while avoiding collisions.

The initial scope of this study was the development of a trajectory planner for two rotary-wing UAVs to follow a moving ground target through a complex three-dimensional (3D) domain. However, the literature revealed a benefit of using a general formulation for multiple UAVs, rather than starting with a single UAV, then expanding it to two UAVs, and then to three and more. This thesis, therefore, presents a trajectory planner for multiple UAVs, as illustrated in Figure 1.2. A ground target (car), is shown moving between obstacles (trees). The UAVs are tasked to follow the moving target, where possible maintaining a specified position relative to the target.

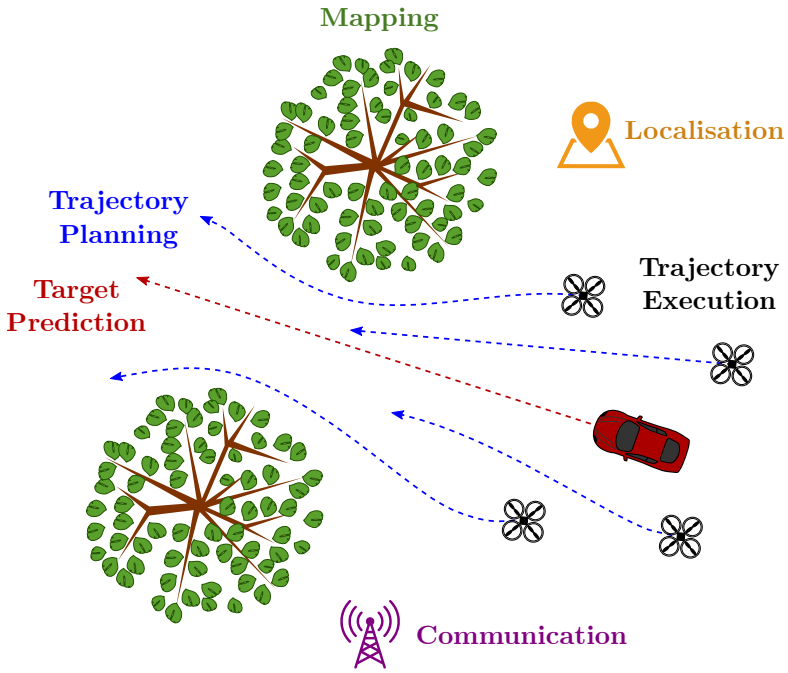


Figure 1.2: A target-following system, where multiple UAVs are tasked to autonomously follow a moving target while avoiding obstacles and one another.

The intended application presents numerous challenges. Firstly, multiple UAVs significantly increase the size of the planning problem, thereby making it difficult to efficiently plan the trajectories. Secondly, the trajectory planner does not know the *actual* future trajectory of the ground target, but performs its planning based on the *predicted* target trajectory. The planner should therefore be able to modify or replan the planned UAV trajectories in real time if the predicted target trajectory changes. Finally, the trajectory planner should take the dynamic constraints of the UAVs into account when planning the trajectories to ensure that the UAVs will be able to execute the planned trajectories.

The thesis utilises many distinct, and to some degree, well-established, research areas to create a real-time trajectory planner. This includes system modelling, control theory, mathematical optimisation, simulation, and numerical methods.

1.2 Target-following system

The trajectory planner is a component within a *target-following system*. Figure 1.3 shows the architecture of a typical target-following system. At the heart of the architecture, and the focus of this thesis, is the *trajectory planner*. However, the trajectory planner does not operate independently. As indicated in Figure 1.3, some components of the target-following system are explored throughout the thesis, as these components impact the design of the trajectory planner.

The planner receives a prediction of the target trajectory and a map of the environment to plan trajectories. The planned trajectories are passed to the trajectory execution component. The trajectory execution component controls the rotary-wing UAVs to execute the planned trajectory using the UAVs' onboard flight control system. Finally, a communication component is required to enable the UAVs to communicate with one another, and to enable communication between the different components of the target following system. These components are briefly discussed in the rest of this section.

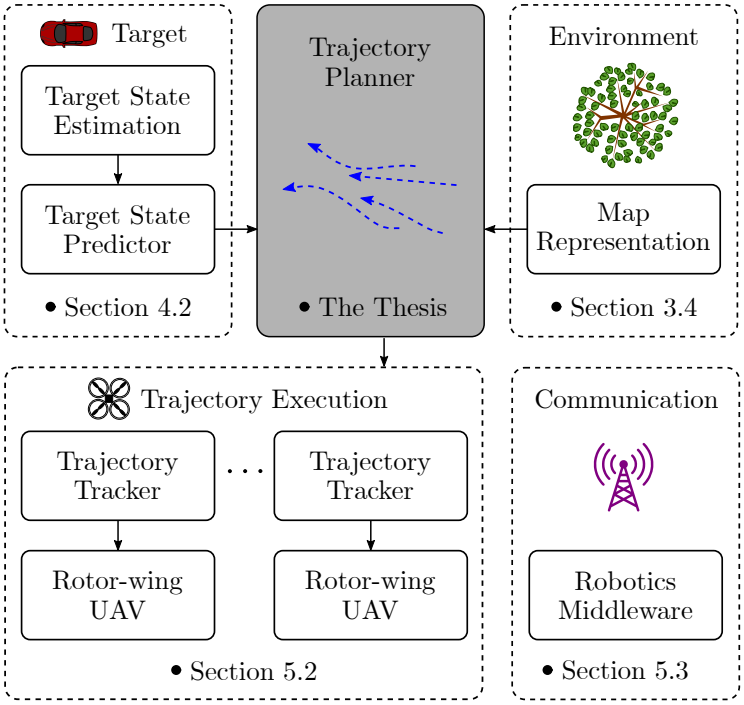


Figure 1.3: Architecture of a target-following system.

1.2.1 Target

The ground target is the subject that is being followed. Common examples include a cyclist or a jogger. It could also be a subject studied by a motion capture system. Implicitly, there is a constraint on the target that it cannot be more agile than, or exceed the maximum speed of, the rotary-wing UAVs following it. As will become apparent later in the thesis, the more agile and the faster the target is, the more taxing the planning problem becomes.

The versatility of a target-following system becomes even more apparent when applications are not limited to following a physical target. By generating an artificial

target, various applications of aerial photography are made possible. By simulating an artificial target through a field, it is possible to survey a terrain with multiple UAVs, or by simulating a virtual leader, formation flight with multiple UAVs can be achieved.

1.2.2 Environment

The UAVs and target operate in an environment which is described by a map. In a natural outdoor environment, obstacles could include trees or bushes. In contrast, in an urban environment, the obstacles are more likely to be man-made such as buildings, towers, and walls. To obtain a map of the environment, a mapping system is needed. Mapping systems often rely on point clouds from depth-sensing cameras (or similar sensors) to build a map of the environment [6].

1.2.3 Trajectory planner

We formulate the target-following problem as having each UAV maintain a desired relative position with respect to the ground target. The desired relative position for a UAV can be parameterised as a desired distance d_{ref}, angle α_{ref}, and height h_{ref} from the ground target, as illustrated in Figure 1.4 (height h not indicated on graph). Note that desired angle α_{ref} is specified relative to the ground target's orientation.

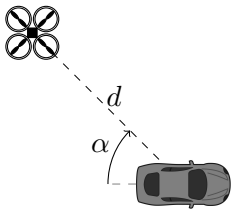


Figure 1.4: Definition of distance d and angle α between the target and the UAV.

A common method to regulate states of a dynamic system is to make use of feedback control, as illustrated in Figure 1.5. The position of the UAV is indicated by $\mathbf{p}$. The *target estimator* calculates the position of the UAV relative to the target. The controller receives the error between the reference position $[d_{\text{ref}}, \alpha_{\text{ref}}, h_{\text{ref}}]^\intercal$ and the current state $[d, \alpha, h]^\intercal$. Based on a predefined control law, the controller applies a control input $\mathbf{u}$ to the plant (UAV) to correct for any deviations.

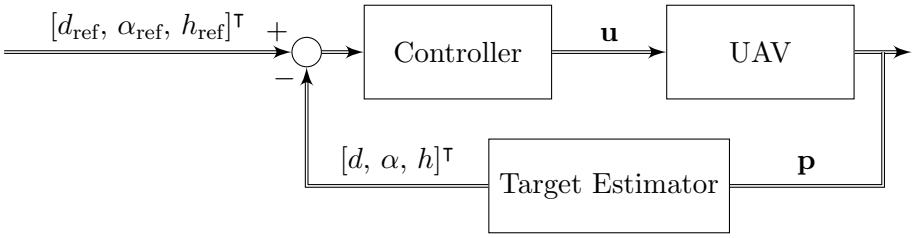


Figure 1.5: Feedback control system to regulate the position of the UAV relative to the target.

When there are no obstacles in the environment, classical control techniques such as Proportional Integral Derivative (PID) controllers could suffice [7]. However, this

approach is problematic when there are obstacles in the environment. If the controller is not aware of the obstacles, it could give a control signal that would result in a collision.

Therefore, a strategy is needed to avoid obstacles. In autonomous systems, the task of planning a trajectory through an environment is known as trajectory planning. Trajectory planning refers to developing a *sequence of actions* for moving a system from an *initial state* to a *goal state* within the *constraints of the system* [8]. The *initial state* is the starting positions of the UAVs. The *goal state* is the desired positions that the UAVs should reach. The *constraints of the system* include the dynamics of the UAVs, as well as the obstacles in the environment.

However, there are some distinct differences between the standard trajectory-planning formulation and target following. In the 'classic' trajectory planning formulation, a goal position is defined, and the task is to plan a trajectory to reach it. For target-following applications, the goal region is of less importance. Instead, it is of more importance to be at a specific reference position relative to the target at each time-step. In that sense, the target-following problem is similar to a control problem.

1.2.4 Trajectory execution

Rotary-wing UAV

A rotary-wing UAV is a type of unmanned aircraft that makes use of spinning rotors to generate lift. Figure 1.6 shows an image of a rotary-wing UAV with six rotors, commonly referred to as a hexacopter. As will become apparent in Section 3.5, the exact configuration of the aircraft is not of much importance for this project. In many cases, the rotary-wing UAV can be abstracted to a point mass from a planning perspective, especially when used with a suitable trajectory tracker, as discussed in the following subsection.

An advantage of rotary-wing UAVs is their ability to generate lift without moving forwards. Their capacity to hover and perform agile manoeuvring makes rotary-wing UAVs well suited for the application of target following.

Figure 1.6: Rotary-wing UAV with six rotors [9].

Trajectory tracker

Once a trajectory has been planned, a UAV needs to execute the trajectory. In the absence of model uncertainty and disturbances, the planned control actions could just have been applied to the UAV. However, in reality, there are many disturbances and uncertainties (e.g. wind disturbances, sensor noise and model inaccuracies). Therefore, a control system is needed to ensure that the trajectory is executed. These types of controllers are commonly referred to as trajectory trackers.

1.2.5 Communication

To perform autonomous navigation for multiple robots (UAVs), the individual robots must continuously communicate with one another and/or a central node to share information such as state and intent, and to coordinate planning.

1.3 Research aim and methodology

The project aim is defined as:

> Develop a trajectory planner for multiple rotary-wing UAVs to cooperatively follow a moving ground target while avoiding collisions with the environment and between UAVs, as well as obeying the dynamic constraints of the UAVs.

The following methodology was used to execute the research project:

1. A literature study was performed to review existing trajectory-planning techniques and to select the technique that would be most suitable for the cooperative target-following application.

2. The cooperative trajectory planner was developed based on the best practices identified in the literature study. A strategy was also developed to modify the planned UAV trajectories to react to a change in the predicted ground target trajectory.

3. The cooperative target following framework was created in the Robotic Operating System (ROS), and the trajectory planner was implemented as a ROS node. Suitable components were selected for the target trajectory predictor, the map of the environment, the trajectory execution guidance system, and the UAV control system.

4. The cooperative target following system was verified in simulation using Gazebo (an open-source 3D robotics simulator). The trajectory planner was first tested on its own, and then as part of the larger target following framework. The UAVs and their flight control system were simulated in Gazebo using a representative UAV model which was developed by Furrer *et al.* [10] and has been validated with flight tests [11].

1.4 State of current research

The current state of robotics research can, to some extent, be compared with a half-completed puzzle. Different research fields have been combined, and a picture is emerging. However, there are still a few pieces missing. Apart from the missing pieces, some pieces look as if they are a fit, but they are not. On the other hand, some pieces do not look like a fit, but they are.

Because the puzzle is partially built, robots have exceeded our wildest expectations to some extent, yet fall short in other respects. A challenging task is getting robots to cooperate on a task. Humans are capable of forming groups to achieve a task, but multi-agent robotics have many opportunities left for improvement.

The goal of multi-agent autonomous robotics would consist of multiple robots autonomously collaborating on tasks, with the ability to anticipate and avoid obstacles (both static and dynamic). A complete general system does not yet exist, but as an active research field, the continued research effort is bringing us closer. Our chosen topic of cooperative target following attempts to create and fit additional pieces to the puzzle of robotics.

In the broader field of robotics, research exists for multiple ground robots to move in a formation on a predefined path through an environment [12]–[14]. In many ways, formation control and target following are quite similar. In both cases, multiple robots (or UAVs) need to move close to each other along a path. However, formation control approaches are not usually designed with the application of following a (physical) target in mind.

For cooperative, multi-UAV target following, several previous research projects have been performed for fixed-wing UAVs, mostly for military surveillance applications. These projects focused on designing paths for fixed-wing UAVs to lower the chance of losing their sight of the ground target [15], [16]. There are, however, significant differences between fixed-wing aircraft, and the more agile rotary-wing UAVs considered in this project. This will be discussed in Section 2.1.2.

Target following with a single rotary-wing UAV is a well-established research field, with the technology even making its way into commercial drones, such as the Skydio UAV [17]. However, research on target following with multiple rotary-wing UAVs (with collision avoidance) is limited and not well documented.

1.5 Primary contributions

The main contribution of this thesis is the formulation of cooperative target following and collision avoidance for multiple rotary-wing UAVs as an optimal control problem. The thesis also describes and motivates the considerations needed to solve the optimal control problem with trajectory optimisation. The design of a trajectory planner for multiple UAVs is presented based on the discussed design considerations. Specific design parameters (planning resolution, planning horizon length and replanning rate) are investigated from both a theoretical and experimental perspective to aid in selecting the parameters for target-following applications.

The trajectory planner is implemented as a Robot Operating System (ROS) node [18], allowing it to be used with different components. The simulation environment is capable of randomly generating obstacles and targets, which can be used to test a planning algorithm rigorously. The project presents not only a trajectory planner, but also a framework on which multiple aspects of autonomous navigation can be tested - this will be highlighted along the way.

The formulation is intended for, but not limited to, rotary-wing UAVs. Although rotary-wing UAVs are studied here, many concepts are transferable to other applications. This includes planning trajectories for fixed-wing aircraft, motor vehicles or aquatic vehicles. Enabling multiple agents to cooperate on a task is an active research field. As such, this project is relevant within the broader autonomous robotics domain. It showcases how optimisation, control theory and trajectory planning can be combined to solve robotics problems, fitting additional puzzle pieces to the robotics domain.

1.6 Research scope

The research scope is defined as follows:

- The focus of the research is the cooperative trajectory planning, and not on the target trajectory prediction, the environment mapping, and the trajectory execution. Suitable components are therefore selected to represent the target trajectory prediction, the environment mapping, and the trajectory execution components.
- The project focuses on rotary-wing UAVs, a type of UAV that produces lift by rotating blades around a fixed mast. These vehicles can hover mid-air, and they are very agile (compared to fixed-wing UAVs), making them well suited for target following.
- Some execution time measurements were performed to provide an indication of whether the algorithm is suitable for real-time implementation. However, the execution timing may still be improved by optimising the software and the target hardware.

The following assumptions are made regarding the information that the trajectory planner can access:

- The trajectory planner has knowledge of the current state (position and velocity) of the target. This assumption is motivated by the fact that various trackers and state estimation approaches exist to estimate the state of a target from aerial footage [19], [20]. However, the future state of the target is not known and is predicted using a prediction model.
- A map of the environment is available, and all obstacles are static. This limits the systems initially to known environments, without any dynamic obstacles. An approach to operating in an unknown environment is to use a trajectory planner which updates its trajectories as more information about the environment becomes available [21]. For this, a fast and reactive trajectory planner is needed. Therefore, planning trajectories quickly in a known static environment is a step towards planning in an unknown dynamic environment.
- The communication between the UAVs is significantly faster than the planning time of the trajectory planner. Communication systems for multi-agent robotic systems have been developed to send messages reliably within a few milliseconds [22]. Therefore, the design of the trajectory planner neglects communication delays. Future projects could use this project to measure the impact of communication delays, and if necessary, include it in the planning formulation.

1.7 Thesis outline

The rest of the report is structured as follows:

Chapter 2: Literature Review

This chapter presents a literature review on target-following approaches, trajectory planning techniques and concepts related to the trajectory planning problem.

Chapter 3: Technique Selection

This chapter makes use of the literature review and identifies optimal control as a good approach to formulate the target-following problem. Design choices are made regarding the trajectory planner, motivated with research on trajectory optimisation approaches.

Chapter 4: Trajectory Planner Design

This chapter presents the design of the trajectory planner. The design can be divided into three categories, namely: (1) the trajectory optimisation strategy, (2) the grid search strategy to provide an initial estimate of the trajectories, and (3) the replanning strategy to plan trajectories in real time. The chapter is concluded with an example to illustrate the trajectory optimisation approach.

Chapter 5: System Implementation

This chapter presents the broader architecture of a target-following system. The chapter highlights the interaction between the trajectory planner and the other components within the target-following system. The simulation environment used to verify the performance of the trajectory planner is also presented.

Chapter 6: Parameter Selection

This chapter presents a guide to measure the impact of, and select, certain design parameters of the trajectory planner. The parameters are the planning resolution, planning horizon length and replanning rate. The experiments are used to select a suitable set of parameters for the current implementation.

Chapter 7: Simulation Results

The chapter verifies the performance of the trajectory planner through simulation. First, examples are presented to illustrate the trajectory planner and target-following system. Secondly, the trajectory planner is tested in a variety of randomly generated scenarios.

Chapter 8: Conclusion

This chapter concludes the report with a summary of the work, the main findings, and opportunities for future research.

Chapter 2

Literature Review

A literature review on target-following research and trajectory-planning techniques is presented in this chapter. First, an overview of current research within the target-following domain is presented. This is followed by an overview of single-agent and multi-agent trajectory planning techniques. Finally, other components within the target-following system are investigated, as these components affect the design of the trajectory planner. This literature review is used extensively in the next chapter for determining a suitable technique for this project.

2.1 Target following

This section expands on the discussion in Section 1.4 by giving an overview of what has been achieved in the target-following domain, as well as related domains such as autonomous UAV navigation and formation flight.

2.1.1 Single UAV target following

Target following with a single rotary-wing UAV is a well-established research field. Some straightforward approaches use only visual feedback information from a camera feed to maintain a distance from a target [7], [23], [24]. More sophisticated systems make use of a prediction of the target trajectory and a map of the environment to plan efficient trajectories [25]–[27].

Visual servoing

One common method is to make use of visual servoing. By making use of image processing, feedback control information are extracted from a camera feed. The feedback controller regulates the position and size of the target within the camera frame. In this way, it regulates its distance and angle relative to the target. Research with visual servoing includes the work of Fonseca and Creixell [7], Nayak and Karaya [23] as well as Rabah *et al.* [24]. However, these projects neglect obstacles in the environment, limiting the use to open environments.

Target following with trajectory planning

Visual servoing alone cannot account for and avoid obstacles in the environment. To avoid collisions, some researchers incorporate trajectory-planning techniques. Woods

and La [25] propose a system based on potential fields methods. Potential field methods are, however, known to get stuck in local minimums in the potential field. Chen and Shen [26] make use of the A* algorithm to map all the free space in the environment. Once the known free space is obtained, quadratic programming is used to generate collision-free trajectories bounded in the free space (this approach will be further discussed in Section 3.4).

2.1.2 Multiple fixed-wing UAVs

For cooperative, multi-UAV target following, several previous research projects have been performed for fixed-wing UAVs, mostly for military surveillance applications. These projects focused on designing paths for fixed-wing UAVs to lower the chance of their losing sight of the ground target [15], [16], [28], [29]. Through constrained optimisation, the loitering paths are adjusted to avoid obstacles in the environment. There are, however, significant differences between fixed-wing aircraft, and the more agile rotary-wing UAVs considered in this project. The dynamics of a fixed-wing UAV are more constrained, being unable to stop abruptly or hover. Therefore, their trajectories are generally planned more conservatively, usually loitering at a distance above any obstacles. The more agile rotary-wing UAVs considered here can follow the target at a closer distance, rapidly manoeuvring through the environment. This increases the need for fast algorithms, swiftly reacting to a ground target changing its course.

2.1.3 Multiple rotary-wing UAVs

Several projects make use of multiple rotary-wing UAVs to track a moving ground target [5], [19], [20]. These projects focus on accurately estimating the state of the target with cameras on multiple UAVs, but neglects obstacles in the environment. Our research is, therefore, complementary to theirs. They focus on accurately estimating the trajectory of a target, given multiple UAVs flying next to the target. Our research focuses on planning collision-free trajectories through a cluttered environment to maintain a position relative to the target. However, research on target following with multiple rotary-wing UAVs (with collision avoidance) is limited and not well documented.

2.1.4 Autonomous UAV navigation

Closely related, although not specifically target following, is the problem of navigating rotary-wing UAVs through a cluttered environment. In this field, UAVs need to navigate through an environment, usually planning for a horizon into the future.

A popular trajectory optimisation based technique is the *Covariant Hamiltonian Optimisation for Motion Planning (CHOMP)* [30]. Originally CHOMP was designed for robot manipulators, but the concept has been extended to Micro Aerial Vehicles (MAVs)[1]. The obstacles in the environment are included as a Euclidean Signed Distance Field (ESDF), which will be described in Section 3.4.1. Inspired by the CHOMP algorithm, Oleynikova *et al.* [21] present a continuous-time trajectory optimisation method for collision avoidance on multi-rotor UAVs. Although the algorithm showed promising results, the success rate of generating a safe trajectory was around 60%, even

[1] a class of miniature UAVs that has a size restriction

with random restarts. Similar to their work is the research of Usenko *et al.* [31], which introduces an efficient way of representing the environment in a 3D circular buffer.

Gao *et al.* [32] expanded the framework of Oleynikova *et al.* [21] by combining the trajectory optimisation with a sampling-based technique (which is further explained in Section 2.2). A rapidly-exploring random graph (RRG) finds a collision-free trajectory through the environment. The trajectory is then used as a starting point for the trajectory optimisation framework, which generates a smooth and dynamically feasible trajectory. This change significantly improved the success rate of the planner, highlighting the importance of a reasonable initial estimate, of the solution, for trajectory optimisation. Our research also employs a search algorithm to provide an initial estimate for a trajectory optimisation algorithm. However, we consider multiple UAVs intending to fly relative to a target. Their focus is on navigating a single UAV to a goal region with an incrementally updating map.

2.1.5 Formation control

Another research domain that is closely related to target following is navigating multiple robots (or UAVs) in a formation through an environment. This is an active research field, both for ground robots in a 2D environment [12]–[14], and rotary-wing UAVs in a 3D environment [33]–[35]. In most of these projects a predefined path is defined which the formation should follow while avoiding static (and in some projects dynamic) obstacles. Although our research does not explicitly deal with dynamic obstacles, following a dynamic target induces a similar effect. A dynamic target in an environment can be viewed, to an extent, as a target standing still with a dynamic, moving environment.

A popular formation control strategy is a leader-following formation [34], [35]. In a leader-following formation, a leader robot (or virtual leader) is selected for which a trajectory is planned. The trajectories of the other robots are then controlled to maintain a position relative to the leader robot. This aligns closely to the objective of our research, following a physical target. However, these projects rarely note cooperative target following as a potential application, focusing more on surveying large areas or cooperative payload transportation [34].

2.1.6 Conclusion

A review of initiatives related to the objective of our study has been given above. The review showed that although research on single UAV applications is common, research on target following with multiple rotary-wing UAVs with obstacle avoidance is not readily available. This project is, therefore, well-positioned to contribute to target following, robotics and autonomous navigation.

The review also revealed two related domains, autonomous UAV navigation in an unknown environment, and formation control with multiple UAVs. These domains are mostly built upon trajectory optimisation and can provide insight into solving our research problem. In the same way, the trajectory planner designed in this thesis can be used to increase the versatility of formation flight systems. Our proposed planner can enable formation-flight systems to quickly react to changes to the path that the formation is following.

2.2 Overview of trajectory-planning techniques

This section discusses different trajectory-planning and control techniques. First, some concepts and definitions are defined, followed by an overview of popular planning techniques.

2.2.1 Concepts and definitions

As background information to the discussion, the following concepts and definitions are introduced:

- **Agent:** The agent refers to the robot (in this project a UAV) which needs to move through the environment. In the case of this project, an agent refers to a rotary-wing UAV.
- **State:** The state of a system is the smallest possible subset of system variables that can represent the entire state of the system at any given time. The state space is the set of all possible states of the system. The state could, for example, represent the position and orientation of an agent [8]. In a multi-agent system, the state could represent the positions and orientations of all the agents in the system.
- **Problem dimensions:** The dimensions of the search problem refer to the dimensions of the state-space. In general, the higher the dimensions, the larger the search space, the more computationally intensive it is to compute the solution.
- **Completeness:** Completeness is a property of a trajectory-planning algorithm. If a method is complete, it means that the algorithm will find the trajectory if it exists.
- **Optimality:** Optimality refers to the quality of the solution. Usually, an objective function is used to define the optimality of a specific path and to compare different trajectories through the environment.

2.2.2 Grid-based search

A popular way to solve trajectory-planning problems is to discretise the state-space into a grid structure, as illustrated in Figure 2.1. This grid can be viewed as a graph, where each gridpoint represents a vertex. Each vertex is connected to the surrounding gridpoints through an edge. Using a graph search technique such as the A* algorithm [36] or Dijkstra's algorithm [37], a search is performed through the environment from the initial position to the goal region (more detail on the A* algorithm is presented in Section 4.6).

Grid-based search can be a very efficient method for low-dimensional problems, but performance degrades as the search region increases. The size of the search space increases rapidly as the number of dimensions increases. This is often referred to as the *curse of dimensionality* [38]. When the dynamics of a system are included in the search problem, the size of the search space increases significantly. For this reason, a grid-based search strategy is not ideally suited for planning with dynamic constraints on the system. However, it is possible to use this algorithm hierarchically [8]. This

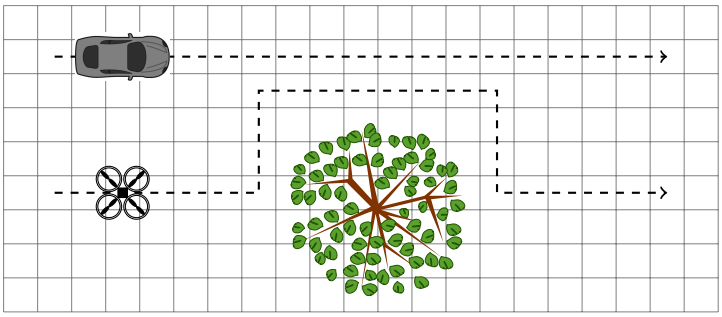


Figure 2.1: Quadcopter following moving target by dividing the area into grids.

involves first, planning a collision-free path while neglecting the dynamics of the system. This path is then smoothed to satisfy the differential constraints of the system. Finally, a feedback control system ensures that the path is executed. This approach reduces the computational complexity of each step. However, some completeness and optimality are sacrificed (completeness and optimality have been defined in Section 2.2.1).

2.2.3 Sampling-based methods

Sampling-based methods were introduced as a response to the 'curse of dimensionality' present in grid-based search methods. Instead of explicitly discretising the configuration space into grids, points are randomly sampled and connected to the original graph with the help of a collision check function.

One popular algorithm is known as the Probabilistic Road Map (PRM) [39], which is illustrated in Figure 2.2. This technique consists of a learning phase as well as a query phase. In the learning phase, the search algorithm randomly samples the configuration space. If it is possible to connect to a previous sample, the new point is added to the graph structure. After the learning phase, the search algorithm enters the query phase. In order to plan a trajectory, the start and goal states are connected to the graph. The path can then be searched with a graph search technique.

This technique works well if the robot operates continuously in the same environment, such as a workshop floor. A graph could be computed in advance for the complete area, and this can be reused if the agent needs to navigate to a new position.

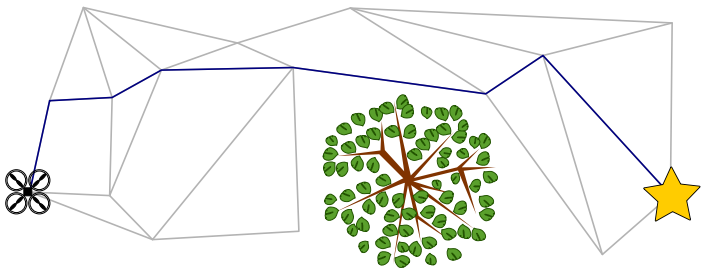


Figure 2.2: Quadcopter planning a path to the goal location making use of a Probabilistic Road Map (PRM).

If the agent is moving through an environment once, it may be unnecessary to generate a graph for the entire environment. Researchers have introduced an iterative sampling-based planner called Rapidly-exploring Random Trees (RRT) [8]. Instead of the learning phase (building a graph of the complete environment) a tree structure is

built, originating from the start region towards the goal region. Such a tree is illustrated in Figure 2.3.

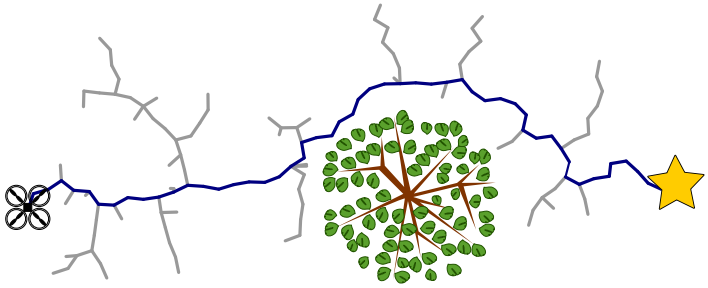


Figure 2.3: Quadcopter planning a path to the goal location making use of Rapidly-exploring Random Trees (RRT).

2.2.4 Artificial potential field methods

Both grid-based search and sampling-based methods aim to capture different configurations in free space in a graph structure. One of the first attempts to directly compute the trajectory was Artificial Potential Fields (APFs) [40]. These techniques create a virtual force field through which a path is planned, as illustrated in Figure 2.4. The goal location emits an attraction force, and the obstacles in the environment emit a repulsive force. This creates a force field hemisphere. The method then performs gradient descent on the potential field to compute a path towards the goal region. The first APF methods were prone to get stuck in local minimums in the potential field. However, various researchers have presented work to overcome this initial limitation [41]–[43].

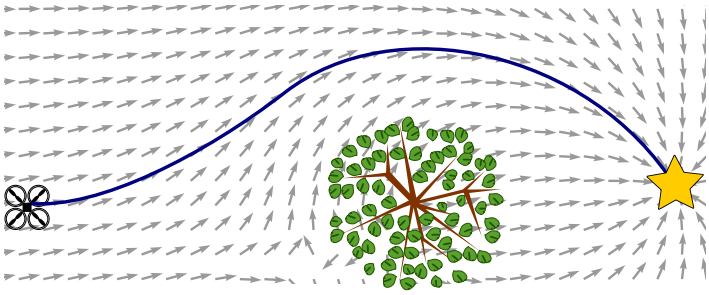


Figure 2.4: Artificial potential field method of planning trajectories.

2.2.5 Optimal control

As stated above, trajectory planning is the task of determining a *sequence of actions* for moving a system from an *initial state* to a *goal state* within the *constraints of the system*. This definition closely aligns with the subject of control theory that deals with the task of determining the set of inputs to control a dynamic system to the desired state. Although traditionally viewed as separate fields, trajectory-planning and control systems are closely related. As computation power increases and the systems that are controlled become more complex, the separation is becoming less distinct. Trajectory-planning techniques could be used to plan the set of controls to reach a

reference state. Similarly, if control problems include an environment, they do not necessarily differ from trajectory-planning problems.

What interleaves the two domains, even more, is that some state-of-the-art control-system techniques are based on viewing the control theory problem as an optimisation problem, known as *optimal control*. A lot of the underlying theory of trajectory planning is also built on optimisation. In both cases, the task problem is formulated as an optimisation problem in the form

$$\begin{aligned} &\underset{\text{input, state}}{\text{minimise (min)}} && \text{path objective} + \text{goal objective} \\ &\text{subjected to (s.t.)} && \text{dynamic constraints} \\ & && \text{path constraints} \\ & && \text{initial state} \\ & && \text{goal state.} \end{aligned} \tag{2.1}$$

The goal is to obtain the *trajectory* and *input signal* to minimise (or maximise) an *objective function*[2] (usually response time or energy consumed), subject to the constraints of the vehicle and constraints in the environment. The trajectory is also subjected to an *initial state* and a *goal state*.

The field of optimal control can be roughly divided into two broad categories, namely *closed-loop solutions* and *open-loop solutions*. The difference is illustrated in Figure 2.5. In the closed-loop formulation (right), the solution is described as a function of the state of the system. There is a policy or control law that describes the solution to reach the goal state from any state. This is desirable, but not feasible for all systems - especially as the dimensions of the state-space increases. In instances where closed-loop solutions are not viable, open-loop solutions (left) are often used. With open-loop solutions, the control input is defined as a function of time. Instead of planning an optimal policy from any state, a single trajectory is planned from an initial state to the goal state. Calculating this single trajectory is generally less expensive than calculating the optimal policy. However, if the robot deviates from the trajectory, replanning the trajectory is necessary. Using open-loop solutions in a feedback configuration is addressed by using a Model Predictive Control (MPC) strategy, which is discussed further in Section 3.7.

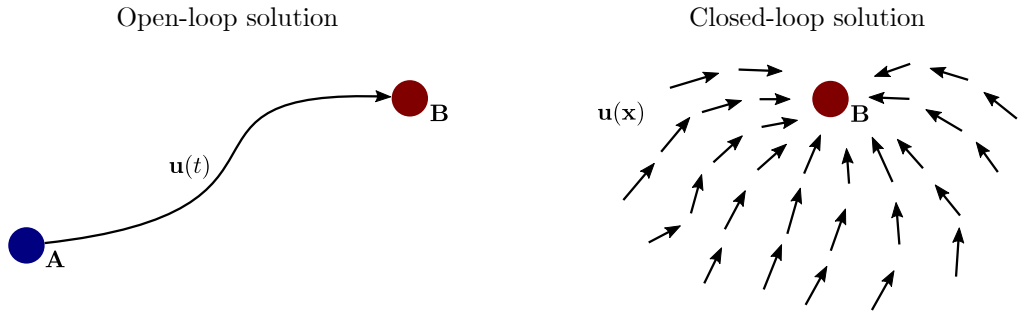


Figure 2.5: A comparison between open-loop and closed-loop optimal control methods [44].

Generating a closed-loop solution for a target-following problem is generally not feasible. If the control policy was determined based on the predicted target trajectory,

[2]The objective function is either a cost function or energy function, which is to be minimised, or a reward function or utility function, which is to be maximised. In this thesis, the objective function is always a cost function, which is to be minimised.

and the predicted target trajectory changes, then the policy becomes invalid. Therefore, the remainder of this thesis only focuses on open-loop methods.

Gradient-based methods

One of the fastest techniques to solve an optimisation problem is to utilise the gradient of the objective and constraint functions. Equivalent to Equation 2.1, consider the optimisation problem in the following form

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{R}^n} \quad & F(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}^L \leq \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{g}^U \\ & \boldsymbol{x}^L \leq \boldsymbol{x} \leq \boldsymbol{x}^U, \end{aligned} \tag{2.2}$$

where F is the objective function and $\boldsymbol{g}$ the constraint function bounded between $\boldsymbol{g}^L$ and $\boldsymbol{g}^U$. The decision variable $\boldsymbol{x}$ is bounded between $\boldsymbol{x}^L$ and $\boldsymbol{x}^U$. To make use of a gradient-based optimisation algorithm, both F and $\boldsymbol{g}$ need to be smooth and differentiable. An optimisation problem in this form, with at least one non-linear function, is known as a Nonlinear Program (NLP). NLPs are common in many research domains, and powerful computer libraries are available to solve these problems. NLP libraries are often based on Newton's method for optimisation, which will be discussed in Section 3.8.

An efficient technique for computing the open-loop solution of an optimal control problem is to transcribe the control problem into an NLP. The general idea of this transcription is to convert the continuous-time dynamics of the system to discrete algebraic equations, commonly referred to as *trajectory optimisation* [45].

Trajectory optimisation is considered to be one of the *state-of-the-art* techniques for solving optimal control problems [46]. However, it is important to note that this approach might not result in the globally optimum solution, as gradient-based optimisation algorithms may converge to a local minimum. The solution is sensitive to the initial guess of the solution, as algorithms iteratively improve the solution, by searching in the direction of the gradient. While this can be partially overcome by using a multi-start approach (restarting the search from a new guess of the solution once a region has been extensively explored), the additional computation cost might be too burdensome. Therefore, it is vital to ensure reasonable estimates for the initial optimisation decision variables.

Gradient-free methods

The gradients of objective and constraint functions are not always available, and in the case of discrete optimisation problems, they do not exist. For this reason, there is a family of gradient-free optimal control methods. In these problems, the optimisation problem is formulated as a *discrete optimisation* problem.

A popular technique for solving this type of problem is mixed-integer programming [47]. This approach also has the advantage of being complete and finding the global minimum of the problem. However, solving the problem is much more expensive computationally than gradient-based methods. Current approaches using mixed-integer programming are too slow for real-time planning. However, as computational power increases and optimisation techniques improve, this approach could become feasible [47].

Although not described as such in Section 2.2.2 and Section 2.2.3, search algorithms such as the A* algorithm could be viewed as gradient-free optimisation. These algorithms search for a trajectory which satisfies the constraints of the problem and minimises an objective function (often distance). Therefore, these algorithms fit the template of Equation 2.1.

2.3 Multi-agent planning techniques

The techniques presented in the preceding section focused on planning trajectories for a single agent. However, this study requires a technique for multiple UAVs. Fortunately, many of the algorithms can be adapted for systems consisting of multiple agents. Two different paradigms, *global planners* and *decoupled planners*, as well as how they are combined in a *hybrid approach* are discussed next [48].

2.3.1 Global approach

For a global approach, a single planner simultaneously computes the trajectories of all the agents, as illustrated in Figure 2.6. The multi-agent system is viewed as a single system consisting of a joint state-space of all the individual agent states. Once this joint state-space is constructed, any of the previously mentioned trajectory-planning techniques can be used.

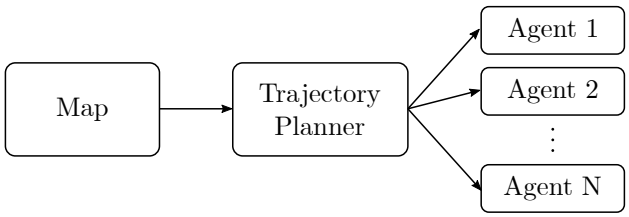


Figure 2.6: Global approach to trajectory planning.

This approach results in a very high-dimensional search space. Therefore, the algorithm of choice should accommodate this. For instance, the A* algorithm is an algorithm that does not scale well in higher dimensions. For the multi-agent, global planner, case, the number of actions in the joint action space is equal to a^n, where a is the number of actions in a set, and n the number of agents. If each agent has six actions, and there are five agents, the number of joint actions at each time-step is $6^5 = 7776$. Therefore, at each node the planning algorithm investigates, it needs to apply 7776 actions. As the number of agents increases, the problem rapidly becomes intractable.

Algorithms with the ability to search in high-dimensional spaces, such as sampling-based methods, perform better. Cap *et al.* [48] designed a pathfinding algorithm based on the RRT* algorithm, called the Multi-Agent RRT*. A joint state is constructed consisting of the states of all the agents. This joint state is sampled using the RRT* algorithm. The study indicated good scalability in terms of the number of agents and grid size in sparse environments. However, the algorithm is still considered to be computationally expensive, especially as the number of agents increases [48].

The advantage of a global approach is that if given enough computational power, it can calculate the optimal solution to the multi-agent problem, such that the agents

do not collide. However, for many applications, this optimal solution comes at an overwhelming computational cost.

2.3.2 Decoupled approach

The other paradigm is to consider a decoupled approach, as illustrated in Figure 2.7. Instead of a single, global trajectory planner, each agent has a separate trajectory planner. The trajectory of each agent is planned independently. After an agent has planned its trajectory, it reserves the position it occupies at each time-step in the *reservation table*. Other agents then consult the reservation table when planning their trajectories. To avoid conflicts when multiple agents plan at the same time-step, a *token-passing strategy* is often enforced. With a *token-passing strategy*, a token is assigned to an agent to plan its trajectory. After planning, the agent reserves its trajectory in the reservation table. The token is then given to the next agent to plan a trajectory.

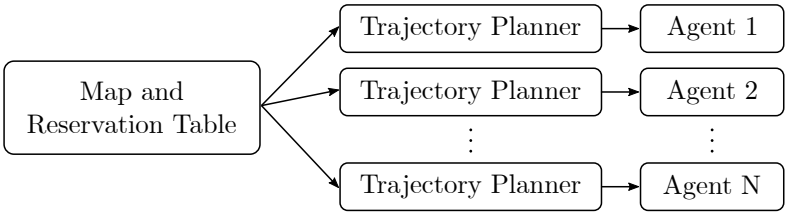


Figure 2.7: Decoupled approach to trajectory planning. Each UAV has a separate trajectory planner to plan trajectories individually.

A popular decoupled approach is the Windowed Hierarchical Cooperative A* (WHCA*) by Silver [49]. First, a path is planned by finding the shortest path for an agent if there were no obstacles in the way. Each agent then takes turns replanning their path, using the optimal path as a true heuristic, and sharing a state-space which includes the positions of the other agents in the environment. However, each agent does not compute the complete path to the goal region, but instead calculates a receding horizon into the future, avoiding any obstacles in the way.

By decoupling the trajectory planning the computational complexity is decreased. This makes it plausible to plan trajectories for systems consisting of many agents. However, it sacrifices optimality and completeness of the trajectories [48]. When an agent plans its trajectory, it only considers what is optimal for its trajectory, and not how its trajectory affects the trajectories of other agents.

2.3.3 Hybrid approaches

A centralised approach has the advantage of generating more optimal paths, at the cost of execution time. A distributed approach sacrifices optimality and completeness but significantly reduces the computational complexity. Researchers have noted this trade-off, and have explored hybrid approaches in an attempt to find a middle ground between the approaches.

One such hybrid approach is the Optimal Anytime (OA) planner by Standley and Korf [50]. Instead of the joint action space of a centralised approach, the OA planner plans an individual path for each agent separately (neglecting the other agents). After

the initial trajectories are planned, sections that result in collisions are identified. A centralised planner is then used to replan areas that result in collisions. The result is paths that are more optimal than that of a distributed approach but which scale better than the centralised approach.

2.4 Related concepts

The trajectory planner is a component within a larger target-following system, where each of the individual components is a research field on its own. The trajectory planner should be capable of interacting with the other components. However, as stated in the project scope (Section 1.6) the focus of the project is the cooperative trajectory planning, and not the target trajectory prediction, the environment mapping, and the trajectory execution. To illustrate how the focus of the project fits into the larger scheme, this section briefly describes the other research fields and the selected components. The architecture of the proposed target following system (which will be described in Chapter 5) is designed to be modular, such that these components can be exchanged for other components if desired.

2.4.1 Trajectory tracking

Planned trajectories are of no use if they cannot be executed. Therefore, this section investigates how planned trajectories can be executed, by making use of a trajectory tracker. Trajectory tracking for UAVs is an active research field. A survey by Lee and Kim [51] cites 79 projects related to UAV trajectory tracking, mostly for quadrotor UAVs.

A method, arguably one of the most intuitive, is to give the desired position at each time-step (in some projects called a 'carrot' [52]) as a reference command to the controllers of the UAV. The controllers try to regulate the error between the UAV position and the 'carrot' to zero. If available, the open-loop control commands to execute the trajectory can be preemptively applied to the actuators of the vehicles. With such an approach, the feed-forward term (applying the open-loop control commands) aids that the UAVs do not lag behind the reference signal, and a feedback controller corrects for any deviations from the planned trajectory.

A more advanced approach is to make use of an optimal control formulation, as introduced in Section 2.2.5. The trajectory tracking problem is formulated as an optimisation problem where the objective is to obtain the input for a trajectory which bests follows the reference trajectory [11], [53]. This closely aligns with the target following objective described in Section 1.2.3, and trajectory tracking is also sometimes referred to as trajectory following. An optimal control formulation also allows for constraints on the states and control input of the vehicle. A popular approach is to make use of trajectory optimisation to calculate an open-loop trajectory, which is repeatedly applied in a Model Predictive Control (MPC) strategy (MPC will be discussed in Section 3.7).

Because many well-researched trajectory trackers exist, this project does not focus on trajectory execution. The focus is on the design of the trajectory planner and its interface to a suitable trajectory tracker for execution. The benefit of this segregated approach is that the trajectory planner presented in this study is not limited to a specific type of UAV or following agent. Another advantage is that from an implementation

perspective, this method of coupling introduces a separation layer to keep critical tasks running despite any failures in the more complex higher-level system [11]. The trajectory tracker is responsible for converting the reference trajectory to actuator commands. Section 5.2.3 describes the design of the selected trajectory tracker designed by Kamel *et al.* [11], and how it integrates with the trajectory planner.

2.4.2 Target state estimation and prediction

In order to follow the ground target, it is necessary to estimate where in the environment the target is, and predict the future state of the ground target. This section elaborates on different approaches.

Target state estimation

A possible approach to estimating the state of the target is to attach sensors, such as Global Positioning System (GPS) and Inertial Measurement Unit (IMU) sensors, directly to the target. However, this is not always ideal or possible, like in the case where the UAVs are following a non-cooperative target. For such cases, researchers have developed techniques to estimate the state of the target from cameras mounted on UAVs.

In both cases, it is usually necessary to use a state estimation technique to estimate the state of the target from the (often noisy) measurements. In general, the internal state of the plant (in this case the state of the target) is estimated by comparing the state measured by the sensors with the predicted state based on a mathematical model of the target [54].

For the design of the trajectory planner, the assumption is made that the position and the velocity of the target are available. This assumption is motivated by the fact that various approaches have been documented to estimate the state (position and velocity) of a ground target from cameras mounted on UAVs [19], [20]. In a complete target-following system, this can either be from a system directly measuring the state of the target, or from an external camera system mounted on the UAVs.

Target state prediction

The trajectory planner designed in this thesis relies not only on the current position and velocity of the target but also on a prediction of the future trajectory of the target. Prediction models estimate the future states of a target by extrapolating from the current and previous states of the target. However, accurately predicting the future state is a difficult task. In the words of Peter Ducker [55]: “Trying to predict the future is like trying to drive down a country road at night with no lights while looking out the back window”. A complex prediction model may predict more accurately, but this is often at the cost of computational power. It is also very unlikely that any prediction model can precisely predict the trajectory of the target. Therefore, it is necessary to replan the trajectories of the UAVs should the target deviate from the predicted trajectories. As predicting the exact future trajectory of the target is not possible, and replanning is inevitable, it is worthwhile to invest time on the planning formulation to ensure that it is as fast as possible.

For the prediction model, this project assumes that the target will continue to travel at its current linear speed and direction. Although this is a rather simple prediction

model, it provides a good starting point for developing a target-following system. If the target-following system can be shown to work on a simplified prediction model, it will likely perform better using a more complex prediction model, if it is available.

Using more complex prediction models is not included in the scope of this project, but for discussion purposes, examples are included here. These include additional second-order motion parameters such as angular velocity and acceleration. It could also include road or path parameters if the target is travelling on a path. The target's future intent can also be used if it is known in advance.

2.4.3 Mapping

To plan trajectories and avoid collisions with obstacles, the trajectory planner needs a suitable map which describes the environment. The task of generating a map of the environment is referred to as mapping. Mapping systems often combine point cloud data from depth-sensing cameras (or similar sensors, such as radars) to build a map of the environment [6]. However, as with the target state estimation problem, there is uncertainty regarding the sensor measurements and the locations of the sensors (mounted on the UAVs). It is challenging to build a map of the environment, as the exact locations of the UAVs are not known. Therefore, the mapping problem is often solved in conjunction with a localisation problem, commonly referred to as Simultaneous Localisation And Mapping (SLAM) [56].

When selecting a map representation for a robotics application, it is important to consider the representation both from a mapping as well as a planning perspective. Some representations are easy to work with from a mapping perspective and can create high-quality maps, but for navigation, fast collision checking and computing distances to obstacles are often of more importance [6]. As discussed in Section 1.6, the project assumes that a complete map of the environment is available. However, selecting a suitable representation in this project could enable future research to expand the system to unknown environments.

This project opts for an occupancy grid representation, one of the most widely used mapping techniques, to represent the environment [57]. With an occupancy grid, the state-space is discretised into a grid structure, as illustrated in Figure 2.8. Each element (called a voxel) represents the probability of that location being occupied with an obstacle. When the map is used for planning, a predefined threshold is used for treating voxels as occupied or not, based on the associated probability.

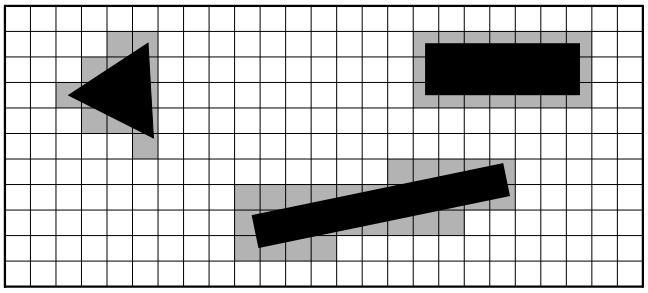


Figure 2.8: Occupancy grid representation of an environment.

As will be discussed in Section 3.4, the trajectory planner makes use of Euclidean Signed Distance Fields (ESDFs) for planning the trajectories. These distance fields

describe the distance of a point to the nearest obstacle. The ESDF is calculated from the occupancy grid representation. For this reason, the exact map representation is not of significant importantance, but rather how efficiently the ESDFs can be built from the representation. Oleynikova *et al.* [58] has developed a mapping system which incrementally builds the ESDF from a Truncated Signed Distance Field (TSDF), a map representation popular for 3D reconstruction. Such a system could be considered in future research but was not investigated in this project.

2.5 Summary

This chapter provided a literature review on target following, trajectory planning, and components within a target following system that may affect the design of the trajectory planner. The literature review is used extensively in the next chapter to determine the best approach for target following. The following key points have been identified from the literature review:

- The project objective is well-positioned to contribute to the domain of target following, autonomous navigation and formation flight.
- The size and complexity of the planning problem significantly increase as the number of agents (UAVs) increases.
- Some planning techniques are better suited to enforce dynamic constraints on the system than others.
- For gradient-based trajectory optimisation, the performance of the algorithm is dependent on the initial estimate (guess) of the solution.

Chapter 3

Technique Selection

The previous chapter provided an overview of possible trajectory-planning techniques for target following. Based on this literature review and guided by the system requirements, this chapter outlines an approach best suited for target following. Optimal control, specifically trajectory optimisation, is selected. However, there are many possible ways to formulate and solve optimal control problems. Therefore, optimal control (specifically trajectory optimisation) is further investigated, and design choices are made with regard to (1) the transcription approach, (2) how to incorporate information about the environment, (3) how to ensure the trajectories are dynamically feasible, (4) the multi-agent planning approach, (5) the strategy for online replanning, and (6) the nonlinear optimisation solver. Where necessary, additional background information is supplied to aid in the technique selection process. The chapter concludes with a summary of how the selected optimisation library works internally, as it is essential for the design of the planner presented in the next chapter.

3.1 Technique requirements

This section determines the desired characteristics of a trajectory-planning algorithm for target following. The original project aim is analysed and converted to a set of technique requirements. Recall the project aim as: *Develop a trajectory planner for multiple rotary-wing UAVs to cooperatively follow a moving ground target while avoiding collisions with the environment and between UAVs, as well as obeying the dynamic constraints of the UAVs.* The project aim can be divided up into the system requirements:

1. Develop a trajectory planner that provides trajectories for multiple UAVs.
2. The UAVs must follow a ground target.
3. The UAVs must avoid collisions with the environment and with one another.

These requirements imply some additional requirements:

1. The planner must be compatible with other components within a target-following system.
2. The trajectory planner should react to changes in the predicted target trajectory.

3. The planned trajectories should be dynamically feasible to ensure that a UAV can execute them.

The first requirement is that the trajectory planner should plan trajectories for *multiple UAVs*. As discussed in the literature review, planning for multiple UAVs significantly increases the size of the planning problem. The size is further increased as the planner should also consider the dynamics of the UAVs. Therefore, the first technique requirement is that the technique should *scale well for high-dimensional problems.*

As stated in Section 1.2.3, in target-following problems, the objective is not to plan a trajectory from an initial position to a goal position. The task is to be at a 'desired' position relative to the target that is followed. For this reason, a suitable technique should *allow an objective function to be minimised along the trajectories.*

The third system requirement states that the trajectory planner should avoid collisions with the environment, as well as collisions between UAVs. This implies that the trajectory planner should be capable of *enforcing constraints along the planned trajectories.* It is something that trajectory search algorithms are capable of, but not necessarily all optimal control techniques (as will become apparent in Section 3.3.2).

The fourth requirement is that the trajectory planner should be compatible with the other components in the target-following system. From an environment representation, this means that *preference should be given to a representation that can be generated in real time.* This would increase the future usability of the planner for operating in an unknown environment.

The final requirement states that the trajectory planner should react to changes in the target trajectory prediction. As the target trajectory is not known in advance, the trajectories should plan real-time trajectories. To achieve this, the trajectory planner should be capable *of quickly replanning trajectories if the target's predicted behaviour changes.*

In summary, the main technique requirements derived from the system requirements are:

- Scale well for high-dimensional planning problems.
- Allow an objective function to be minimised along the trajectories.
- Be capable of enforcing constraints along the planned trajectories.
- A map representation that can be generated in real time is preferred.
- The planner should quickly replan trajectories if the target's predicted behaviour changes.

3.2 Planning technique

From the planning techniques discussed in Section 2.2, optimal control is the technique that most naturally fits the technique requirements listed above. An optimal control formulation allows for both an objective function and constraints. In order to implement this, the target-following problem was formulated to fit the structure of an optimal control problem. The target-following problem can be expressed as:

$$\begin{aligned} &\underset{\text{input, state}}{\text{minimise (min)}} && \text{target following error,} \\ &\text{subject to (s.t.)} && \text{dynamic constraints of the UAVs,} \\ & && \text{a minimum distance between UAVs,} \\ & && \text{a minimum distance from obstacles,} \\ & && \text{the initial state of the UAVs.} \end{aligned} \tag{3.1}$$

As identified by the requirements, the method is to perform fast and reliably, even when applied to high-dimensional problems. As mentioned in the literature review, the NLP class of optimisation methods is well suited to meet these requirements. However, because the optimal control problem is not natively formulated as an NLP, problem transcription is required. These are presented and discussed in the next section.

3.3 Transcription approach

Transcription is the approach of converting the original optimal control problem into an Nonlinear Problem (NLP). In an attempt to organise the techniques, transcription methods are often divided into different categories; most notably, *direct methods vs indirect methods*, as well as *shooting methods vs collocation methods*. It is important to note that, as Betts [59] states in a review paper, the introduced divisions are a means of categorisation. Not every technique falls neatly into one category or another. However, understanding the different categories helps to determine which techniques are best suited for the intended application.

3.3.1 Indirect methods vs direct methods

One approach to categorising the techniques is to distinguish between *indirect methods* and *direct methods*. With indirect methods, the necessary conditions for optimality are calculated. Those conditions are then discretised, and root solving is used to find the solution. Indirect methods tend to calculate very accurate solutions, but it is harder to formulate and solve the problem than with direct methods [45]. In contrast, with direct methods, the planning problem is discretised to a set of algebraic equations. This discretised problem is then solved with a numerical optimisation technique. With direct methods the problems tend to be easier to pose and to solve [60]. However, this is often at the expense of a loss of accuracy.

Traditionally, indirect methods are preferred in the aerospace domain, where they are used to calculate the optimal trajectory for a vehicle travelling to or from space. In these cases, accuracy is preferred above planning time, as planning could happen offline before the mission occurs. Direct methods are popular in the robotics and real-time manipulation community where planning time is prioritised [60]. For the application of target following a direct approach would be preferred, as keeping the implementation fast and efficient is beneficial for real-time planning.

3.3.2 Shooting methods vs collocation methods

Another division in trajectory planning is between *shooting methods* and *collocation methods*. Shooting methods are based on simulation and make use of an explicit

integration scheme. Collocation methods are based on function approximations and make use of an implicit integration scheme.

Shooting methods

Figure 3.1 presents an example to illustrate how shooting methods approach trajectory optimisation. The position of the UAV is described by $\mathbf{p} = [x, z]^\intercal$. The task is to obtain the input $\mathbf{u}(t) = [\ddot{x}(t), \ddot{z}(t)]^\intercal$, that allows the quadrotor UAV to reach the goal area by accelerating as little as possible. The trajectory of the UAV $\mathbf{x}(t)$ is described by its position and the velocity, $\mathbf{x}(t) = [x(t), \dot{x}(t), z(t), \dot{z}(t)]^\intercal$.

Shooting methods start with an initial input $\mathbf{u}(t)$, and simulates the dynamic equations describing the system to generate the trajectory $\mathbf{x}(t)$. The resulting trajectory is measured with the objective function, and the input is readjusted. Starting with the new input, the process is repeated until the optimal trajectory is determined. Because the simulation process is repeated, Figure 3.1 shows multiple trajectories from the UAV to the goal region, each corresponding to a different iteration.

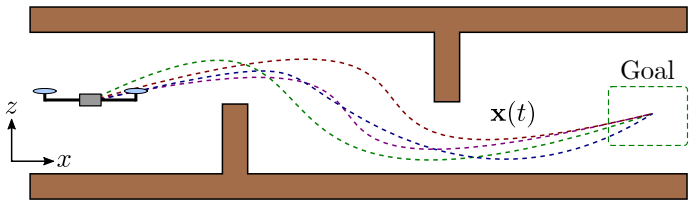


Figure 3.1: Shooting approach for solving a trajectory optimisation problem.

The system dynamics is often described by a function $\boldsymbol{f}$, which describes the change in state for the system in the form

$$\dot{\mathbf{x}}(t) = \boldsymbol{f}(t, \mathbf{x}(t), \mathbf{u}(t)). \tag{3.2}$$

The state of the system is obtained by numerically integrating $\boldsymbol{f}$, from the initial time-step t_0 to the final time-step t_f, as

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \boldsymbol{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \\ \int_{t_0}^{t_f} \dot{\mathbf{x}}(\tau)\mathrm{d}\tau &= \int_{t_0}^{t_f} \boldsymbol{f}(\tau, \mathbf{x}(\tau), \mathbf{u}(\tau))\,\mathrm{d}\tau \\ \mathbf{x}(t_f) &= \int_{t_0}^{t_f} \boldsymbol{f}(\tau, \mathbf{x}(\tau), \mathbf{u}(\tau))\,\mathrm{d}\tau + \mathbf{x}(t_0) \end{aligned} \tag{3.3}$$

An important note is that only the control inputs are part of the optimisation decision variables, and the trajectory is determined through an external integration component. This makes it possible to enforce constraints on the input, but challenging to enforce constraints on the states of the trajectories.

Collocation methods

In contrast to shooting methods which uses an explicit integration scheme, collocation methods rely on function approximations and enforce the integration implicitly. Figure 3.2 presents the same scenario as Figure 3.1 to illustrate collocation methods. The idea is to take the continuous-time functions and convert them to a discrete set of

real numbers. This is achieved by creating a set of $N+1$ gridpoints, at a resolution of t_s. The continuous-time variables and functions are sampled at each gridpoint. The states and inputs become a set of discrete states in time. These gridpoints become the decision variables of the optimisation algorithm. For $N+1$ gridpoints, the set becomes

$$\begin{aligned} t &\rightarrow [t_0,\ t_1,\ \dots\ ,\ t_N] \\ \mathbf{x}(t) &\rightarrow [\mathbf{x}_0,\ \mathbf{x}_1,\ \dots\ ,\ \mathbf{x}_N] \\ \mathbf{u}(t) &\rightarrow [\mathbf{u}_0,\ \mathbf{u}_1,\ \dots\ ,\ \mathbf{u}_N]. \end{aligned} \tag{3.4}$$

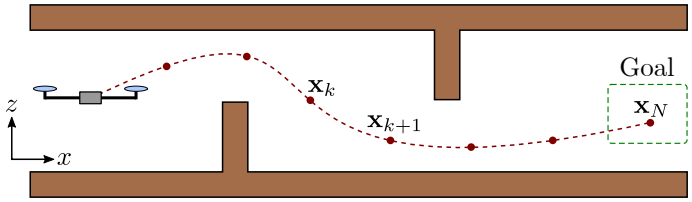


Figure 3.2: Collocation approach for solving a trajectory optimisation problem.

It is still necessary to ensure that the trajectory satisfies the dynamic constraints of the system. This is achieved by approximating the dynamics between gridpoints, which can be achieved in various ways [44]. A popular approximation is to assume that the dynamics vary linearly between gridpoints and is referred to as trapezoidal collocation. With the approximation, the dynamics can be approximated as

$$\dot{\mathbf{x}}(t) = \boldsymbol{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \tag{3.5}$$

$$\int_{t_k}^{t_{k+1}} \dot{\mathbf{x}}(\tau)\mathrm{d}\tau = \int_{t_k}^{t_{k+1}} \boldsymbol{f}(\tau, \mathbf{x}(\tau), \mathbf{u}(\tau))\,\mathrm{d}\tau \tag{3.6}$$

$$\mathbf{x}_{k+1} - \mathbf{x}_k \approx \frac{t_s}{2}(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k) \tag{3.7}$$

where $\boldsymbol{f}_k = \boldsymbol{f}(t_k, \mathbf{x}_k, \mathbf{u}_k)$ is the system dynamics at knot point k. Equation 3.7 is enforced for all gridpoints along the trajectory.

With collocation methods, the optimisation decision variables include both the control inputs as well as the discretised states. Because the states are part of the decision variables, it is easier to enforce constraints along the trajectory, than with shooting methods. Collocation methods are also better suited for systems with complex control inputs. The drawback of collocation methods is that they tend to be less accurate than shooting methods, due to the function approximations.

Summary

Both shooting methods and collocation methods should be considered. However, on implementation level with the handling of obstacle constraints, (as will be discussed in Section 3.4.2) collocation has some inherent features that make it more attractive for the target-following implementation [44].

3.4 Incorporating obstacles

As stated in Section 2.4.3, a method is needed to incorporate the environment into the trajectory optimisation problem. This section selects a suitable map representation as well as a method to incorporate the representation into the optimisation problem.

3.4.1 Map representation

Gradient-based trajectory optimisation generally needs a representation of the environment that is smooth and differentiable. There are various well established approaches to represent the environment in the optimisation problem. This subsection highligts three approaches and selects a suitable technique.

Geometric representations

One way (initially used in this study) is to represent obstacles in the environment with geometric shapes such as polygons. For instance, obstacles can be made up of simpler shapes such as cylinders, cubes or pyramids. Some advanced methods make use of regression techniques to fit high-dimensional polynomial surfaces over the obstacles [61]. The advantages of this technique are that it is easy to work with from a planning perspective. However, this requires that the polygons are fitted onto the map. Usually, this approach is troublesome for complex geometries and real-time online mapping as it is quite computationally expensive.

Bound solutions to free space

Another strategy is to make use of a search algorithm to map out all the free space in the environment. One such algorithm is IRIS (Iterative Regional Inflation by Semi-definite programming), which is illustrated in Figure 3.3 [47]. By searching through the environment, large convex regions without obstacles are identified. The optimisation algorithm then constrains the trajectories to remain within the outer boundary of these convex regions. The advantage of describing the free space is that it can result in a simpler optimisation problem to solve, especially in comparison with approaches where the obstacles are individually described [62]. However, obtaining the free space regions of the map is also quite computationally expensive.

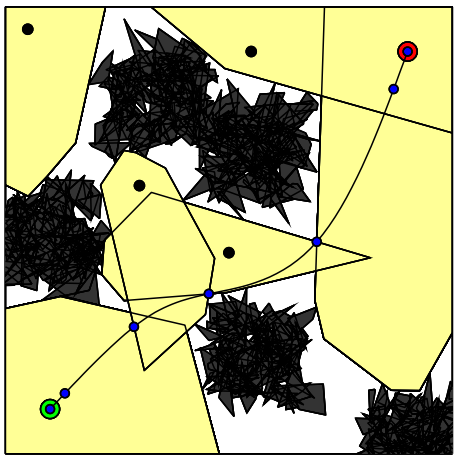


Figure 3.3: IRIS algorithm for computing obstacle free regions within a cluttered map [47].

Euclidean Signed Distance Fields (ESDF)

Euclidean Signed Distance Field (ESDF) is a potential field which gives the distance to the nearest obstacle in the environment. Such a field is demonstrated in Figure 3.4. A

potential field is constructed where the value corresponds to the distance to the edge of the nearest obstacle. Negative numbers represent regions inside the obstacles, and positive numbers represent regions outside of obstacles.

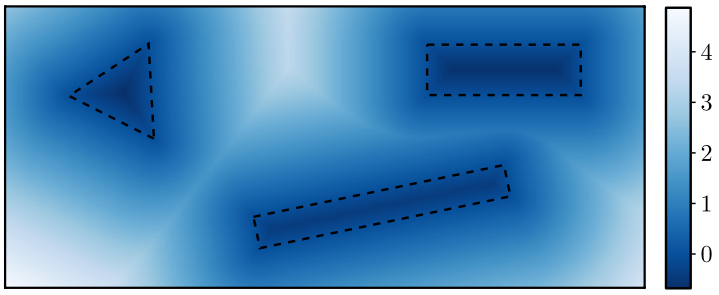


Figure 3.4: Euclidean signed distance field of the environment.

This representation has the advantage of providing a smooth description of the environment. Generating ESDFs can be a computationally expensive task and used to be a bottleneck in performing trajectory optimisation online [30]. Early research that made use of ESDFs had to build the complete map offline before it could be used [30], [63]. Since then, various projects have developed techniques to build approximate signed distance fields incrementally from point-clouds. Libraries such as *Voxblox* [58] and *Fiesta* (Fast Incremental Euclidean diSTAnce fields) [64] are capable of incorporating new information into an original ESDF within a couple of milliseconds.

Summary

Above three approaches to represent the environment within a trajectory optimisation problem have been presented. For this project, ESDFs are selected. Due to the existence of fast techniques to build ESDFs incrementally online, future projects could extend the trajectory planner to operate in unknown environments. ESDFs are also capable of describing arbitrary complex static obstacles, not limiting the trajectory planner to maps consisting of basic geometric shapes.

3.4.2 Obstacle constraints in trajectory optimisation

In general, an optimisation problem can enforce constraints in two ways. The first is to enforce the constraints as *hard constraints*. With hard constraints, there is no flexibility. A solution will not be obtained if the constraints cannot be satisfied. The second approach, *soft constraints*, offer a more relaxed alternative. Soft constraints typically enforce constraints as part of the objective function, for example, adding a scaled value of the constraint violation to the objective function. This allows the optimisation algorithm to compromise on some of the constraints should it drastically decrease the objective function.

In general, soft constraints are well suited for optimisation problems where the penalty of constraint violations are not severe. These might include product design or budget targets, where the gains in the objective function might outweigh the penalty of constraint violation. In contrast to this, hard constraints might be preferred for optimisation problems where a constraint violation leads to a catastrophic failure of the system. The disadvantage of this selection is that it limits the choice of optimisation algorithms to those that are able to treat constraints. Viable computationally efficient

constrained optimisation algorithms might not always be available. As a result, problems where the use of hard constraints is desirable are often reformulated to use soft constraints, even if constraint violations will lead to catastrophic failures.

In this project, it is desirable to formulate the obstacles as hard constraints. This is because there is little benefit to allowing any constraint violation, as this will lead to a collision and possible damage to the vehicle. While this limits the choice of optimisation algorithms, the collocation techniques presented previously are capable of enforcing constraints along a path, while remaining computationally efficient. Therefore, this project selects hard constraints to enforce obstacle constraints.

3.5 Enforcing dynamics

The trajectory planner must consider the dynamics of the UAVs that execute the trajectories. However, Mellinger and Kumar [65] have shown that it is not necessary to consider the full dynamics of the UAV explicitly. Their research has shown that a quadrotor UAV is a differentially flat system. This means the entire state of the system can be expressed as a function of the x, y, z position of the vehicle centre of mass and the yaw angle and their derivatives. Ferrin *et al.* [66] derived a representation for a general multi-rotor case.

The ability to represent the system as a differentially flat system has significant implications from a trajectory-planning perspective as it is possible to plan trajectories in terms of the position and the yaw angle of the vehicle. If the trajectories are smooth with sufficiently bounded derivatives, the UAV should be capable of executing the trajectory.

A similar argument for abstraction is often made from a control theory point of view. If the inner-loop dynamics of the UAV are significantly faster than the outer-loop dynamics, it is possible to argue that the inner-loop dynamics may be neglected.

This project opts for a similar approach. The trajectories are planned in terms of the differentially flat inputs. However, the trajectory planner should at least consider second-order dynamics, to ensure the derivatives of the states are sufficiently bounded.

3.6 Global planner vs decoupled planner

As indicated in the literature review (Section 2.3), the use of a global planner leads to more optimal trajectories than the use of a decoupled approach. However, a decoupled approach offers better scalability in terms of the number of agents. Modern NLP solvers, such as IPOPT [67], are capable of efficiently solving problems with a high number of decision variables. This is achieved by exploiting the structure of the matrices, as well as calculating only the local solution to the problem. Therefore, a centralised (global) planner could be feasible.

This project implements a global approach where a single trajectory planner plans trajectories for all agents. The advantage of first formulating the global planner case is that it provides insight into the scaling and complexity of the multi-agent robotics problem. This could provide a benchmark for future projects to investigate a decoupled approach, both in terms of trajectory optimality and execution time.

3.7 Replanning

A strategy is required to replan trajectories should the target deviate from its predicted trajectory. A well-established technique in control theory for using trajectory optimisation in a closed-loop manner is Model Predictive Control (MPC). The idea is that if trajectory optimisation can be performed quickly enough, it can be used as the feedback policy. The general idea is: (1) measure the current state, (2) optimise the trajectory from the current state, (3) execute the first action planned through trajectory optimisation, (4) let the dynamics evolve for one time-step and repeat [68].

Although initially designed for control systems, the concept of MPC has been successfully applied to target following [16], [28], [29]. In these projects, the trajectory of the target is predicted. The predicted trajectory is used to plan trajectories for the UAV to follow the target. This project makes use of a similar approach.

Model Predictive Control (MPC)

MPC is an optimal control strategy which plans for a receding horizon into the future. The strategy is illustrated in Figure 3.5. The current time-step is indicated by k. A desired reference trajectory for the system is indicated. A trajectory optimisation technique plans the optimal inputs for a horizon, $k + p$, into the future, where p is known as the planning horizon length.

The controller then applies the first input in the control sequence for a single sampling time. At the next time-step, $k + 1$, the process is repeated. The prediction horizon and the optimal inputs are planned up to $(k + 1) + p$. As before, an input is determined by making use of trajectory optimisation. This process is repeated for the duration of the trajectory execution. The time between replanning, t_{MPC}, is referred to in this thesis as the replanning interval (which is usually significantly shorter than the planning horizon). The tempo of replanning is called the replanning rate, which is defined as $f_{\text{MPC}} = \frac{1}{t_{\text{MPC}}}$.

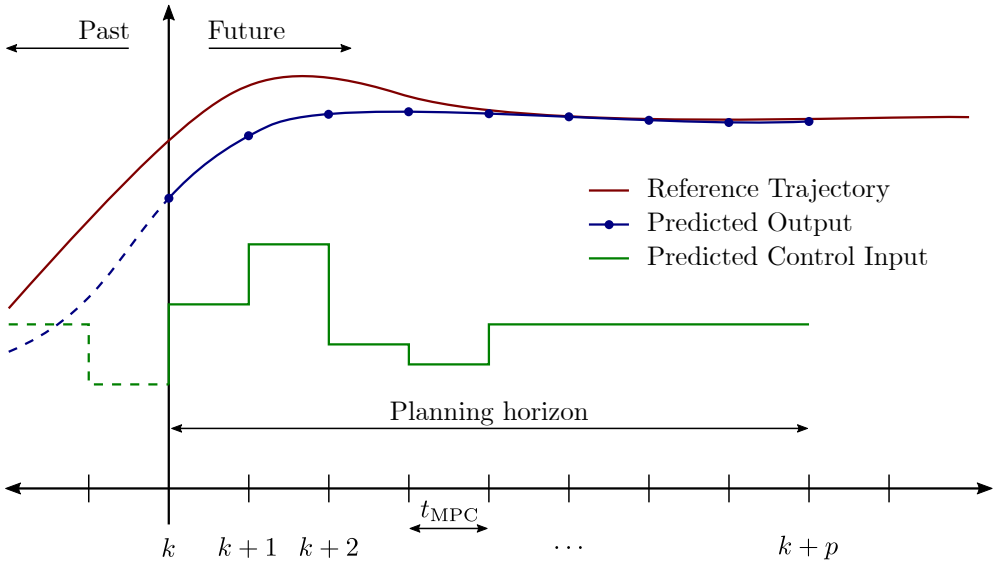


Figure 3.5: Receding horizon control strategy of MPC controller (adapted from Behrendt [69]).

Generally, there is a trade-off between the length of the planning horizon and the

replanning interval. Both a fast replanning rate and a longer planning horizon increase the quality of the solution. However, the longer the planning horizon, the more complex the optimisation process, which limits the maximum replanning rate. This trade-off is further explored in Chapter 6.

3.8 NLP Libraries

A non-linear optimisation algorithm is needed to solve the transcribed trajectory optimisation problem. Traditionally, solving full non-linear optimisation problems was a time-consuming process, limiting trajectory optimisation to offline applications. Through advances in optimisation techniques and computational power, solving NLPs quickly and efficiently has become attainable. Much research has gone into designing modern, state-of-the-art solvers. Table 3.1 provides a list of some popular NLP solvers. Some of the listed solvers require a commercial licence, whereas others are available through the Eclipse Public License (EPL). Most of these solvers make use of either an interior-point method or Sequential Quadratic Programming (SQP).

Table 3.1: Summary of popular NLP solvers, adapted from [70]

Solver	License	Method
IPOPT [67]	Open-source (EPL)	Interior-point method
BONMIN [71]	Open-source (EPL)	Interior-point method
KNITRO [72]	Commercial	Interior-point method
WORHP [73]	Commercial	Interior-point method , SQP
SNOPT [74]	Commercial	SQP
Fmincon [75]	Commercial	Interior-point method , SQP

NLPs resulting from direct collocation tend to have a large number of decision variables, as the decision variables include both the control input and the discretised state. However, when calculating the derivatives of objective and constraint functions (termed the Jacobian matrix), there are many zero elements within the matrix. Such a matrix is often called a sparse matrix. Some solvers are specifically designed to exploit the sparsity structures within the optimisation problem. One such solver is IPOPT. The efficiency of IPOPT, combined with its permissive open-source licence, makes it an excellent choice for direct collocation optimisation problems. For these reasons, IPOPT is selected as the NLP solver for this project.

3.9 Mathematics of solving NLPs

It may be tempting to decouple the transcription process from the optimisation process and treat the optimisation algorithm as a black box. However, in reality, the transcription and optimisation processes are tightly coupled. It would be challenging to understand and subsequently debug the transcription process without an understanding of the underlying techniques of the optimisation solver. This section provides an overview of the main concepts behind IPOPT, which is necessary for understanding some of the design choices. The explanations in this section are combined from a variety of sources, most notably Wächter and Biegler [67] and Betts [45].

3.9.1 Newton's method for optimisation

As an introduction to non-linear programming, this section starts with a simple unconstrained problem, and shows how it can be solved with Newton's method for optimisation. Although most solvers are built upon the highly developed field of mathematical optimisation, the basic principles of Newton's method are still relevant [45]. The goal is to obtain the optimal solution $\boldsymbol{x}^*$ that minimises the objective function F, as illustrated in Figure 3.6. The algorithm requires an initial guess of the solution, indicated by $\boldsymbol{x}_0$. Mathematically, the optimisation problem is expressed as

$$\min_{\boldsymbol{x}\in\mathcal{R}^n} \quad F(\boldsymbol{x}). \tag{3.8}$$

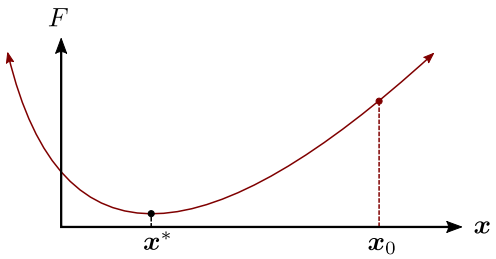


Figure 3.6: Objective function with single optimisation variable.

Newton's method works by making a second-order approximation of the objective function and minimising it. This approximation is achieved by making use of Taylor's theorem. By making use of Taylor's series expansion, the objective function $F(\boldsymbol{x})$ can be approximated around the initial guess of the solution as

$$F(\boldsymbol{x}) \approx F(\boldsymbol{x}_0) + \nabla_{\boldsymbol{x}} F(\boldsymbol{x}_0)(\boldsymbol{x}-\boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_0)\nabla^2_{\boldsymbol{xx}} F(\boldsymbol{x}_0)(\boldsymbol{x}-\boldsymbol{x}_0), \tag{3.9}$$

where $\nabla_{\boldsymbol{x}} F(\boldsymbol{x}_0)$ is the derivative of $F(\boldsymbol{x}_0)$ (called the Jacobian matrix), and $\nabla^2_{\boldsymbol{xx}} F(\boldsymbol{x}_0)$ the second (called the Hessian matrix), at the initial guess $\boldsymbol{x}_0$.

A quadratic approximation is selected due to it being the simplest approximation that does have a minimum point [45]. This minimum point $\bar{\boldsymbol{x}}$ can be obtained by setting the derivative of the expansion equal to zero. Specifically

$$\frac{dF}{d\bar{\boldsymbol{x}}} \equiv \nabla_{\boldsymbol{x}} F(\bar{\boldsymbol{x}}) = \boldsymbol{0} = \nabla_{\boldsymbol{x}} F(\boldsymbol{x}_0) + \nabla^2_{\boldsymbol{xx}} F(\boldsymbol{x}_0)(\bar{\boldsymbol{x}}-\boldsymbol{x}_0). \tag{3.10}$$

Solving for the new point yields

$$\bar{\boldsymbol{x}} = \boldsymbol{x}_0 - [\nabla^2_{\boldsymbol{xx}} F(\boldsymbol{x}_0)]^{-1}\nabla_{\boldsymbol{x}} F(\boldsymbol{x}_0). \tag{3.11}$$

The minimum of the approximation, $\bar{\boldsymbol{x}}$, is used as the starting point of the next iteration. The process is repeated as illustrated in Figure 3.7. As before, the point is approximated by a quadratic function, and the minimum point is computed. As shown on the figure, as the algorithm progresses, the update steps become smaller. This happens because the derivative of the objective function approaches zero as the algorithm converges. Finally, the process terminates when the changes between two iterations are smaller than a predetermined threshold.

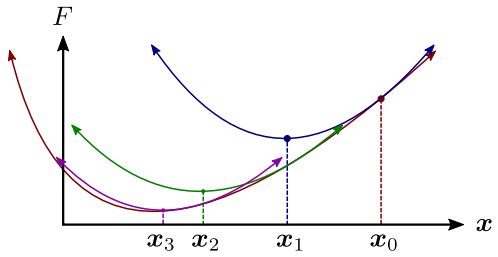


Figure 3.7: Objective function with 2nd order approximation.

Newton's method illustrates a few key concepts. Firstly, the optimisation algorithm makes use of an iterative process to reach the solution. It also shows that the closer the algorithm is initialised to the solution, the fewer iterations are needed to solve the problem. Secondly, the example shows the reliance of the algorithm on the first and second derivative of the problem. The objective function should be smooth, such that the derivative is defined everywhere.

The example also showed that Newton's method approximates the objective function at every iteration as a quadratic equation. Therefore, the more the objective function resembles a quadratic function, the faster the algorithm will converge to a solution. Figure 3.8 illustrates two objective functions to convey this concept. The first objective function (left) will offer good convergence, as it approximates a quadratic function very well. The second objective function (right) will provide poor convergence. The objective function has very steep gradients at some places, with a very shallow bowl near the minimum.

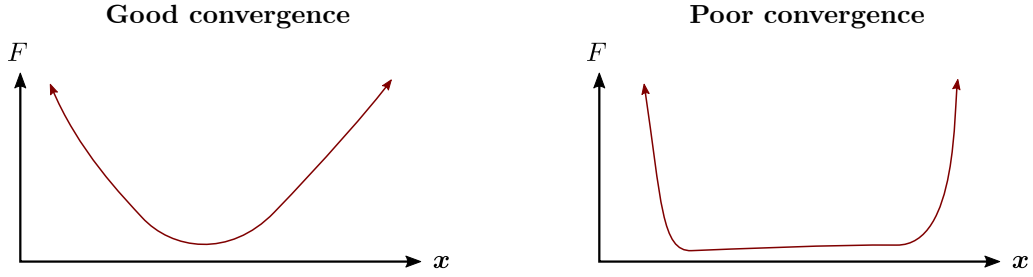


Figure 3.8: Comparison of objective functions in terms of convergence.

The poor convergence example is often the result when there are many possible solutions that satisfy (or near) the minimum [60]. For the target-following problem in this project, there may be more than one trajectory that results in the same optimal value for the objective function. The design presented in Chapter 4 solves the problem by including the energy (acceleration) needed to execute the trajectory within the objective function. The addition of a small regularization term to the objective function (such as the integral of the control squared along the trajectory) puts a shallow bowl in the objective function, forcing a unique solution [60]. Another advantage is that the resulting trajectories are generally smoother and requires less energy to execute than when the regularization term is neglected.

Discussions in earlier sections noted that the optimisation algorithm only finds the local solution to the optimisation problem, and not the global solution. For further clarification, Figure 3.9 presents an objective function with two minimums, $\boldsymbol{x}_1^*$ and $\boldsymbol{x}_2^*$. The minimum $\boldsymbol{x}_2^*$ is the lowest value on the entire function and it is therefore called the global minimum. Although minimum $\boldsymbol{x}_1^*$ is higher than $\boldsymbol{x}_2^*$, it is the lowest value

in the surrounding area. It is therefore called a local minimum. The solution that the optimisation algorithm will reach depends on what region the initial estimate starts. This figure shows why a good initial estimate is vital. Chapter 4 investigates the use of a grid search strategy to provide a suitable starting point of the solution for the optimisation algorithm.

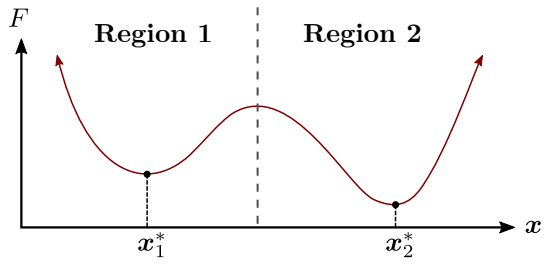


Figure 3.9: Local minima and global minima of the optimisation problem.

3.9.2 Constrained optimisation

This section illustrates how (equality) constraints can be enforced on an NLP with the Lagrange multiplier technique. The optimisation problem considered here is given by Equation 3.12, where $F(\boldsymbol{x})$ is the objective function and $\boldsymbol{h}(\boldsymbol{x})$ is the newly introduced equality constraint.

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{R}^n} \quad & F(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0} \end{aligned} \tag{3.12}$$

Figure 3.10 presents an example for two decision variables $\boldsymbol{x} = [x_1, x_2]^\intercal$, showing the contour lines for both the objective function F and the constraint function $\boldsymbol{h} = \boldsymbol{0}$. The idea behind the Lagrange multiplier technique is to find the point where the contour lines of the objective function and the constraint function are tangent to each other [45]. This is achieved by introducing a new function, called the Lagrangian $\mathcal{L}$, defined as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = F(\boldsymbol{x}) - \boldsymbol{\lambda}^\intercal \boldsymbol{h}(\boldsymbol{x}) \tag{3.13}$$

where $\boldsymbol{\lambda}$ is a vector called the Lagrange multiplier.

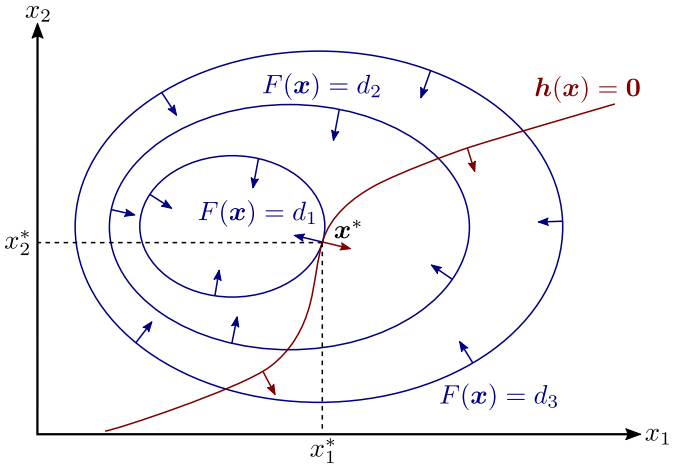


Figure 3.10: Lagrange multiplier method for constrained optimisation [76].

The solution $\boldsymbol{x}^*$ is constrained to $\boldsymbol{h}(\boldsymbol{x}) = 0$, which means the solution should fall somewhere on the contour line $\boldsymbol{h}(\boldsymbol{x}) = 0$. If you 'walk' along this contour line, in the direction such that F descends, and find the point where F is not changing, you have reached a local minimum. If the value can change by 'walking' further along the contour, it means the minimum has not been reached yet. The point where the value of F does not change is where the contour lines are tangential to each other.

This point can be calculated by setting the gradient of the Lagrangian equal to a zero vector $\nabla\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \mathbf{0}$. The equation, together with the necessary conditions to ensure that it is indeed a minimum, is solved with a numerical root-finding method. These conditions, along with a generalisation for inequality constraints, is further discussed in Appendix A.

These examples illustrated some of the basic concepts of obtaining solutions to NLPs. In reality, modern solvers are built on advanced numerical methods to rapidly solve optimisation problems. However, all the principles introduced in this section remains relevant. A more in-depth discussion on optimisation is presented Appendix A. This includes strategies to improve the convergence of the algorithms (called globalisation strategies), as well as how gradients of functions are calculated with a technique called Automatic Differentiation (AD), through a software library called CasADi [70].

3.10 Summary

This chapter applied the concepts introduced in the literature review to identify a suitable trajectory-planning technique for target following. An optimal control formulation was selected. Additionally, component design choices were also made, considering their suitability for the target-following problem. In summary, the following design choices have been made in this chapter with regards to trajectory planning:

- **Planning technique**: An optimal control formulation was selected for the planning technique, as it allows for a natural representation of the target-following problem.
- **Optimal control strategy**: A trajectory optimisation strategy was selected, where the target-following problem is transcribed to an NLP. The transcribed NLP will be solved with a gradient-based optimisation library.
- **Transcription approach**: A direct collocation method was selected to transcribe the target-following problem. Direct collocation is suitable for enforcing hard constraints along trajectories.
- **Incorporating obstacles**: Obstacles are represented in the optimisation problem through ESDFs. ESDFs provides a smooth representation of the environment, and can describe obstacles consisting of complex shapes. The obstacle constraints are also enforced as hard constraints.
- **System dynamics**: The system dynamics are enforced with trapezoidal collocation, and the trajectories are planned in terms of the differentially flat inputs of a multi-rotor UAVs.
- **Multi-agent planning**: To plan trajectories for multiple UAVs, a single global planner was selected.

- **Replanning strategy**: A replanning strategy similar to a Model Predictive Control (MPC) strategy was selected. The trajectory planner continuously replans for a receding horizon into the future.
- **NLP Solver**: IPOPT was selected as the optimisation library of choice due to its ability to solve large, sparse optimisation problems quickly.

Chapter 4

Trajectory Planner Design

This chapter presents the design of the trajectory planner. First, an overview of the planning strategy, along with the proposed architecture, is presented. This is followed by the modelling of the system, the transcription process for trajectory optimisation, the search strategy for providing an initial estimate, and the replanning strategy to update the trajectories in real time. Finally, the chapter is concluded with an example to illustrate how trajectory optimisation solves the trajectory planning problem.

4.1 Overview

The trajectory planning process is summarised in Figure 4.1. The first step is to predict the trajectory of the target with a linear prediction model. The second step is to use grid-based searches performed in parallel to find initial solutions for the individual UAV trajectories. These trajectories are used to initialise the trajectory optimisation process with a good initial starting point of the solution. The grid-based searches plan

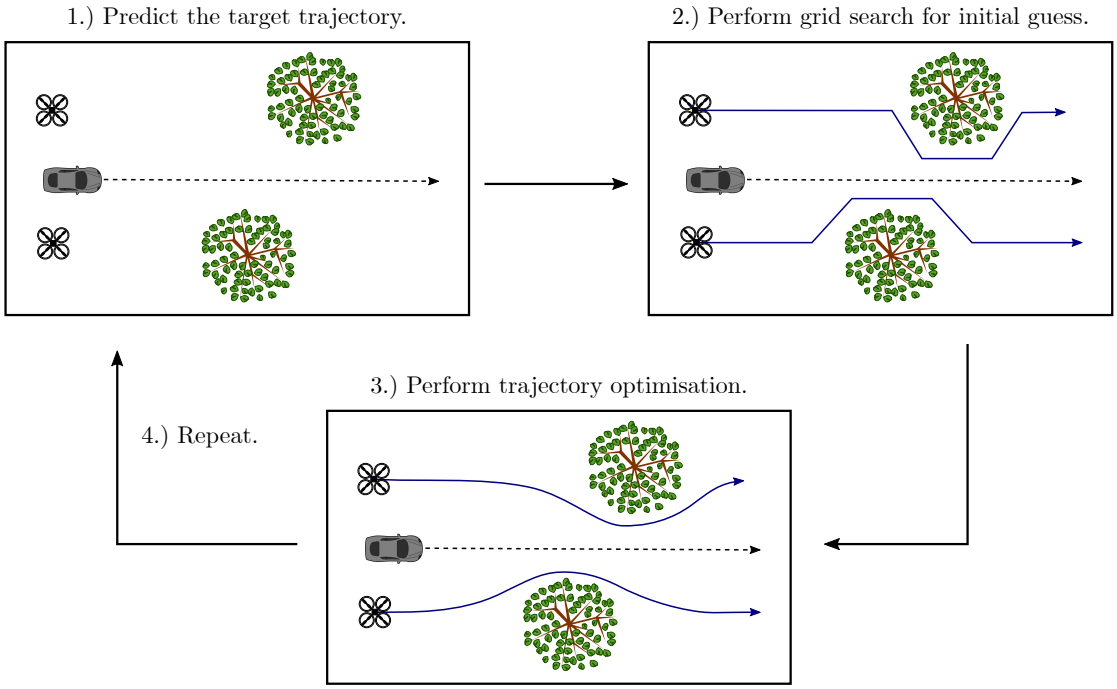


Figure 4.1: Summary of trajectory planning process.

the individual trajectories for the UAVs around static obstacles in the environment but do not check for collisions with other UAVs. The grid-based searches do not take the dynamics of the UAVs into account when planning the initial trajectories. The third step is to perform trajectory optimisation by iteratively improving the UAV trajectories, starting from the initial estimate provided by the grid-based searches, but now taking into account the dynamic constraints of the UAVs and checking for collisions between the UAVs and for collisions with static obstacles in the environment. The trajectory optimiser formulates and solves the target following problem as a centralised constrained optimisation problem. The objective function is designed to minimise the difference between the UAVs' actual positions and their ideal reference positions relative to the target at every time step. The UAV dynamic constraints are included as equality constraints and the collision avoidance constraints are included as inequality constraints in the formulation of the optimisation problem. This trajectory planning cycle is repeated at a fixed replanning rate for a fixed planning window into the future using a receding horizon similar to Model Predictive Control (MPC). At each iteration, the trajectory of the target is predicted and new trajectories are planned.

4.2 Trajectory planner architecture

A block diagram of the proposed trajectory planner is shown in Figure 4.2. The trajectory planner receives a prediction of the target's future trajectory. As introduced above, the planner is fundamentally based on optimal control, which makes use of trajectory optimisation to transcribe the planning problem into a Nonlinear Program (NLP). The NLP is passed to the NLP solver along with an initial estimate of the solution. The NLP solver iteratively improves the estimate of the solution until the (local) optimal trajectory is obtained. The solution is interpolated to convert the discreet points to smooth splines. The next step is to calculate the different yaw angles of the UAVs along the trajectories. (The yaw angle is planned separately from the position trajectory, as will be motivated in Section 4.3.2). Finally, the trajectory planner is implemented within a replanning strategy to react to changes in the predicted target trajectory. The next section describes the models that are used for the target, the UAVs, and the static obstacles. The sections after that present the design of each of the components shown in Figure 4.2.

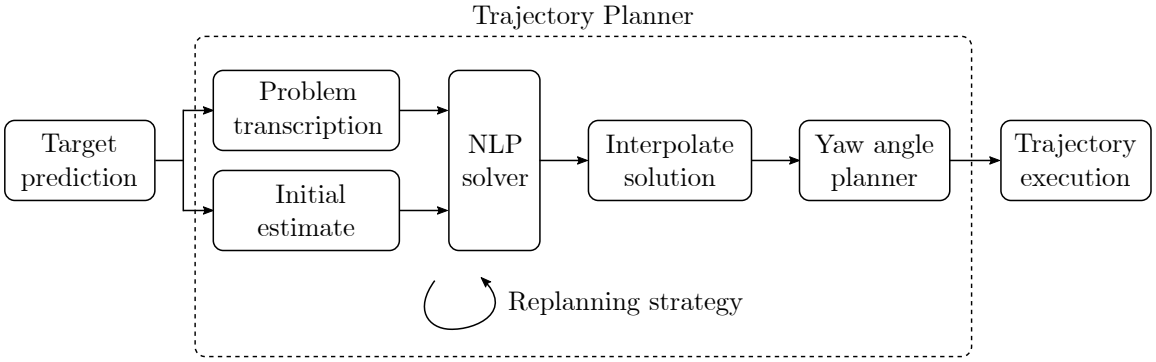


Figure 4.2: Architecture of the proposed trajectory planner.

4.3 Modelling

This section describes the models that are used for the ground vehicle (target), the rotary-wing UAVs, and the environment.

4.3.1 Ground vehicle (Target)

Figure 4.3 shows the target vehicle coordinated in the world axis system. The state of the target is described by its position in the world frame, $\mathbf{t} = [x, y]^\intercal$, its heading ψ, and its forward speed v. The state of the target is therefore represented by its state vector

$$\mathbf{x}_{\text{target}} = [x, y, \psi, v]^\intercal. \tag{4.1}$$

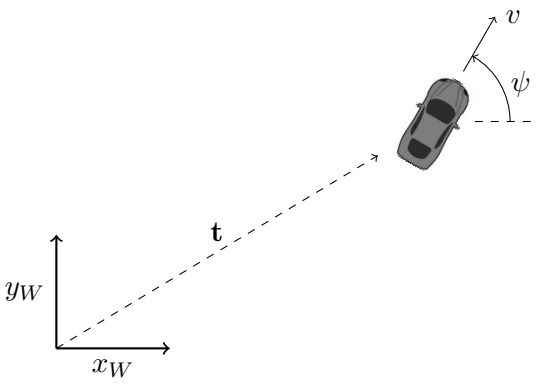


Figure 4.3: Coordinate frame of the target.

Target trajectory prediction model

The predicted target trajectory is obtained by using a constant velocity prediction model. The prediction model assumes that the target will continue to travel at its current velocity at a fixed heading. The predicted position $\hat{\mathbf{t}}(t)$ of the target at a time-step t is calculated as

$$\hat{\mathbf{t}}(t) = \mathbf{t}_0 + v_0 \begin{bmatrix} \sin(\psi_0)t \\ \cos(\psi_0)t \\ 0 \end{bmatrix} \tag{4.2}$$

where $\mathbf{t}_0$ is the current position, v_0 is the current velocity, and ψ_0 is the current heading of the target vehicle.

4.3.2 Rotary-wing UAVs

Figure 4.4 shows the two UAVs within the world frame. Due to the differential flatness property (as described in Section 3.5), it is sufficient to describe a state of a rotary-wing UAV as a function of its position, $\mathbf{p} = [x, y, z]^\intercal$, its heading ψ, and their time derivatives $\dot{\mathbf{p}} = [\dot{x}, \dot{y}, \dot{z}]^\intercal$ and $\dot{\psi}$ [65].

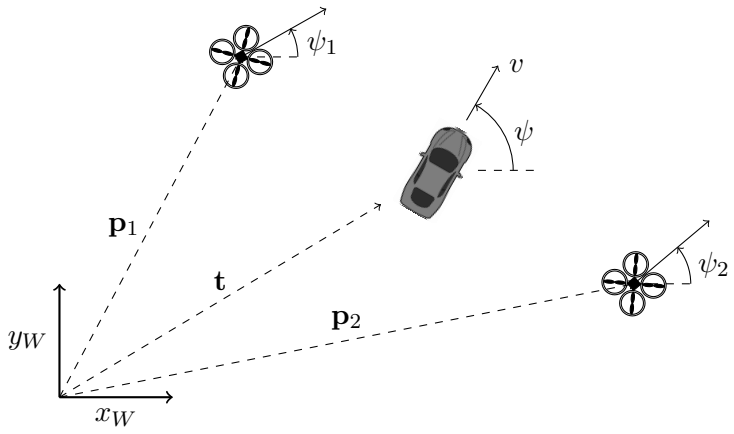


Figure 4.4: Coordinate frames of the target and the UAVs in the world frame.

The yaw angle does not affect the position of the UAV in the world, and it does not impact collisions between UAVs. Therefore, for this project, the design choice was made to separate the yaw planning from the translational planning. Trajectory optimisation is used to generate collision-free trajectories. After the collision-free trajectories have been planned, the desired yaw angles are calculated (more detail will be presented in Section 4.8).

This decoupling decreases the computational complexity of the problem, but in some instances, it may introduce a loss of optimality. More optimal trajectories may be possible if the yaw angles are considered when planning the paths. Future projects could design a trajectory planner that incorporates the yaw angle.

It is also important that the positions and the velocities of the UAVs should be smooth so that the trajectories are dynamically feasible. To achieve smooth trajectories, at least a second-order representation is needed. Therefore, the state of an individual UAV is defined as

$$\begin{aligned} \mathbf{x} &= [\mathbf{p}, \dot{\mathbf{p}}]^\intercal \\ &= [x, y, z, \dot{x}, \dot{y}, \dot{z}]^\intercal, \end{aligned} \tag{4.3}$$

with the input defined as

$$\begin{aligned} \mathbf{u} &= [\ddot{\mathbf{p}}]^\intercal \\ &= [\ddot{x}, \ddot{y}, \ddot{z}]^\intercal. \end{aligned} \tag{4.4}$$

4.3.3 Obstacle representation

As motivated in Section 3.4, Euclidean Signed Distance Fields (ESDFs) is selected to represent the obstacles in the environment. In this project, the ESDF is defined as a scalar distance field $o(\mathbf{p})$. For each point in space $\mathbf{p}$, the ESDF returns a scalar value which is the distance to the nearest obstacle from that point.

4.4 Optimisation problem formulation

This section defines the decision variables used, the objective function and constraints for the optimisation problem.

4.4.1 Decision variables

The decision variables $\boldsymbol{x}$ is the set of variables that the optimisation algorithm can adjust in order to minimise the objective function. For a direct collocation technique, as described in Section 3.3.2, the decision variables include both the state trajectory and the input signal. The trajectories of all the UAVs are discretised into collocation points. The trajectory is sampled into $N-1$ segments, each at a constant sampling time t_s. The sampling time is also referred to as the planning resolution. Figure 4.5 illustrates the decision variables for two UAVs with $N = 9$ collocation points. Formally, the decision variables for A agents with N collocation points are expressed as

$$\begin{aligned}\boldsymbol{x} = [&\mathbf{p}_1^0, \ldots, \mathbf{p}_1^{N-1},\ \dot{\mathbf{p}}_1^0, \ldots, \dot{\mathbf{p}}_1^{N-1},\ \ddot{\mathbf{p}}_1^0, \ldots, \ddot{\mathbf{p}}_1^{N-1},\ \ldots,\\ &\mathbf{p}_A^0, \ldots, \mathbf{p}_A^{N-1},\ \dot{\mathbf{p}}_A^0, \ldots, \dot{\mathbf{p}}_A^{N-1},\ \ddot{\mathbf{p}}_A^0, \ldots, \ddot{\mathbf{p}}_A^{N-1}]^\intercal.\end{aligned} \tag{4.5}$$

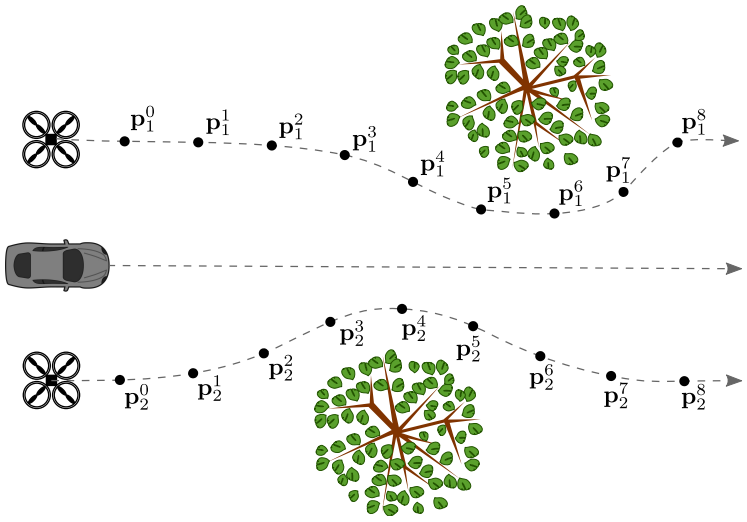


Figure 4.5: Decision variables for two agents where the trajectories are sampled into 9 collocation points.

4.4.2 Objective function

The objective function is selected to minimise the deviation of the UAVs' positions from their ideal reference positions relative to the target at each sampling instant. Figure 4.6 shows the ideal position for a UAV relative to the target. The ideal position is defined by a reference distance d, a reference angle α, and a reference height h (not indicated).

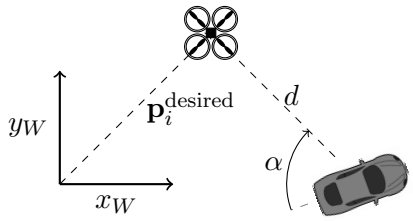


Figure 4.6: Definition of distance d and angle α between the target and the UAV.

Calculating the deviations in d and α is a relatively expensive operation. A less computationally expensive cost function is to define the desired position vector $\mathbf{p}^{\text{desired}}$,

calculated from the ideal distance, angle and height. An objective function representing this deviation is then defined as

$$F(\boldsymbol{x}) = \sum_{i=1}^{A} \sum_{k=0}^{N-1} (||\mathbf{p}_i^{\text{desired}}(t_k) - \mathbf{p}_i(t_k)||^2) \tag{4.6}$$

where $\mathbf{p}_i^{\text{desired}}(t)$ is the desired position of agent i at time t, and $\mathbf{p}_i(t)$ the actual position at the same time.

As described in Section 3.9.1, penalising the control signal of the trajectory within the objective function adds shallow bowl to the objective function, which helps the optimisation algorithm converge to a single solution [60]. Additionally, ignoring the acceleration of the UAVs in the objective function results in unnecessarily aggressive acceleration by the UAVs, limiting their total flight time. Therefore, the objective function in Equation 4.6 is expanded to penalise aggressive acceleration. The final objective function is given by

$$F(\boldsymbol{x}) = \sum_{i=1}^{A} \sum_{k=0}^{N-1} (||\mathbf{p}_i^{\text{desired}}(t_k) - \mathbf{p}_i(t_k)||^2 + w||\ddot{\mathbf{p}}_i(t_k)||^2) \tag{4.7}$$

where $\ddot{\mathbf{p}}_i(t_k)$ is the acceleration of agent i at time-step t_k and w is a weight which can be adjusted based on the desired response. When a smaller weight w is chosen, the trajectories use more aggressive accelerations and are less smooth; when a larger weight w is chosen, the trajectories use less aggressive accelerations and are more smooth.

4.4.3 Constraints

The formulation makes use of boundary constraints, equality constraints, and inequality constraints to formulate the trajectory optimisation problem. The constraints are divided into two categories. The first type is *collision constraints*, which ensure that the trajectories are collision-free. The second type is *dynamic constraints*, which ensure that the trajectories are dynamically feasible.

Collision constraints

A minimum distance is enforced between UAVs, and between the UAVs and any obstacles. These constraints are modelled as inequality constraints. Figure 4.7 illustrates the different constraints enforced on the system. The distances o_i represents the distance between agent i and its nearest obstacle. This distance is obtained from the ESDF. The distance c_{ij} represents the distance between agent i and agent j. For example, the inequality constraint for the scenario shown in Figure 4.7 is represented by

$$[o_1, o_2, o_3, o_4, c_{12}, c_{13}, c_{14}, c_{23}, c_{24}, c_{34}] \geq \boldsymbol{d}_{\text{min}} \tag{4.8}$$

where $\boldsymbol{d}_{\text{min}}$ is a vector containing the minimum distances allowed for each constraint. These constraints are enforced at all the time-steps.

The distance between UAVs can be enforced in more than one way. For example, a minimum distance between agent i and agent j can be expressed by both Equation 4.9 and Equation 4.10. However, the optimisation algorithm converges faster when the constraint is represented by Equation 4.10 rather than Equation 4.9. This is because

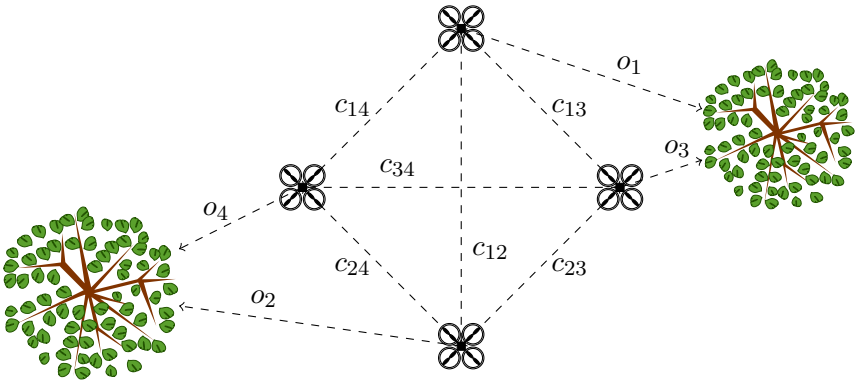


Figure 4.7: Constraints between the obstacles and UAVs.

the gradient of Equation 4.10 remains roughly within the same order of magnitude, irrespective of whether the UAVs are close to or far away from one another. It was, therefore, decided to use the distance constraint represented by Equation 4.10.

$$(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \geq d_{\min}^2 \tag{4.9}$$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \geq d_{\min} \tag{4.10}$$

However, the gradient of a square-root function is undefined when its argument equals zero. When two UAVs are initialised at the same position at a time-step, the optimisation algorithm will not be able to calculate the gradient. Therefore, Equation 4.10 is expanded to

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 + \epsilon} \geq d_{\min} \tag{4.11}$$

where ϵ is a small positive number ($\epsilon \approx 0.01$). The value of ϵ should be small enough not to significantly impact the constraint, yet large enough to provide numerical stability.

Dynamic constraints

Equality constraints are used to enforce the dynamics of the system at each collocation point. The dynamic constraints are enforced with trapezoidal collocation. The dynamics are approximated as

$$\begin{aligned} \dot{\mathbf{x}} - \int \boldsymbol{f}(\mathbf{x}(t), \mathbf{u}(t))\, \mathrm{d}t = 0 \\ \dot{\mathbf{x}}(t_k) + \frac{t_s}{2}(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k) - \dot{\mathbf{x}}(t_{k+1}) \approx 0 \end{aligned} \tag{4.12}$$

where $\mathbf{x}_k$ and $\mathbf{u}_k$ are the state and input at the current time step, $\mathbf{x}_{k+1}$ is the state at the next time step, and t_s is the sampling period. The collocation method approximates the change as linear between collocation points.

The optimisation solver allows for constraints to bound the decision variables between an upper and lower boundary. Boundary constraints are enforced on the velocity and acceleration decision variables to ensure that the trajectory is dynamically feasible.

4.5 NLP Solver

The formulated NLP, along with an estimate of the solution for the planning problem (which will be discussed in Section 4.6) are given to the selected NLP solver, IPOPT. Similar to Newton's method described in Section 3.9, IPOPT determines a step direction to improve the trajectory by looking at the gradients of the objective and constraint functions. Using an iterative process, the initial estimate of the solution is improved until the optimal solution is obtained. IPOPT returns the optimal set of decision variables, which needs to be interpolated to obtain the optimised trajectories. This will be described in Section 4.7. More detail regarding the NLP solver is presented in Appendix A.

4.6 Initial estimate

IPOPT only calculates the local optimal solution to the optimisation problem. Therefore, it is essential to initialise the solver with a reasonable initial estimate of the decision variables. The convergence rate of the solver is highly dependent on the initial estimate of the decision variables. There is a trade-off between investing more time generating a good estimate, to enable the solver to converge more quickly to a solution, or by saving time by generating a naive estimate quickly, but then having the solver converge more slowly to the solution.

For an understanding of the importance of a suitable initial estimate, it is necessary to understand the ESDF and the corresponding gradient field. To illustrate the problem, a 2D environment is presented in Figure 4.8 with two possible initial estimates. The one initial estimate goes through the obstacle, whereas the other initial estimate goes around the obstacle.

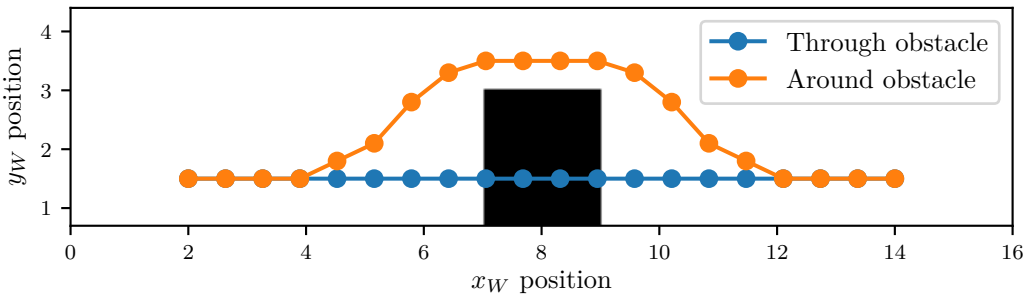


Figure 4.8: Two strategies for initialising the optimisation algorithm.

Figure 4.9 and Figure 4.10 shows the corresponding ESDF, as well as the gradient of the ESDF along the trajectories. In the first case, shown in Figure 4.9, the path is initialised through the obstacle. To satisfy the obstacle constraints, the optimiser will try to move the path out of the obstacle area. The optimiser will shift the points along the gradient of the ESDF. However, the gradient of the ESDF is zero along the y-axis. The optimiser will be able to satisfy the collision constraints, but the dynamic constraints will be infeasible.

In Figure 4.10, the optimiser is initialised around the edge of the obstacle. The optimisation algorithm can optimise the trajectory without moving the collocation points into a collision area. (The gradient of the ESDF is always perpendicular to the

edges of the obstacle, as the edge represents a contour line) This example illustrates why a reasonable initial estimate is so important for trajectory optimisation.

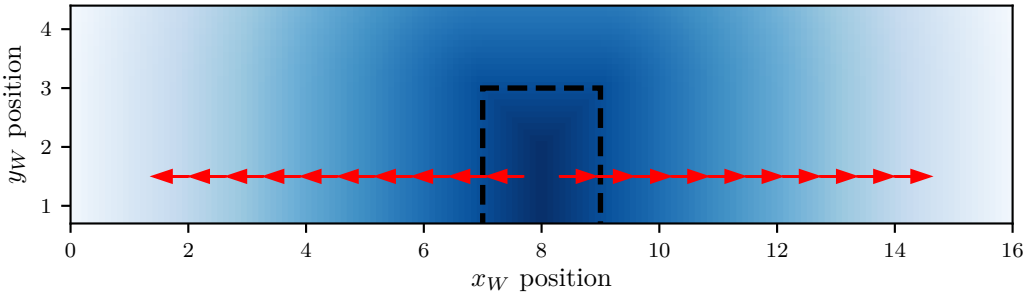


Figure 4.9: Initial estimate going through obstacle.

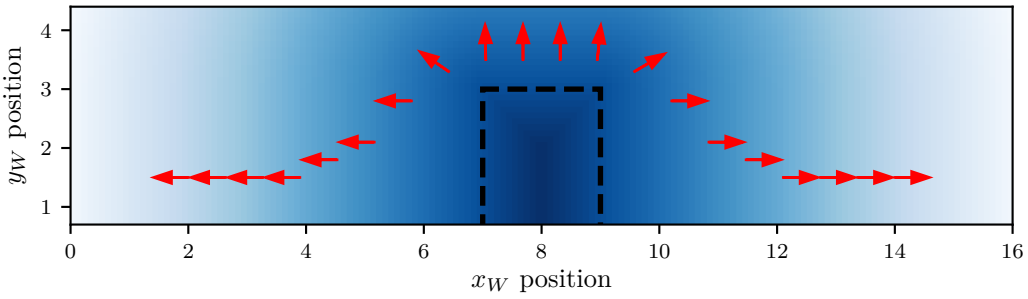


Figure 4.10: Initial estimate going around obstacle.

For this project, it was decided to perform grid-based searches using the A* algorithm to obtain initial estimates for the individual UAV trajectories. The grid-based searches plan the individual trajectories for the UAVs around static obstacles in the environment. However, it does not check for collisions with other UAVs. The dynamics of the vehicles are neglected to keep the search space of the algorithm as small as possible. The advantage of neglecting collisions between UAVs is that the searches can be performed in parallel.

A decentralised planner usually relies on a token allocation system with a reservation table. For this to be possible, the agents plan their trajectories sequentially, as illustrated in Figure 4.11.

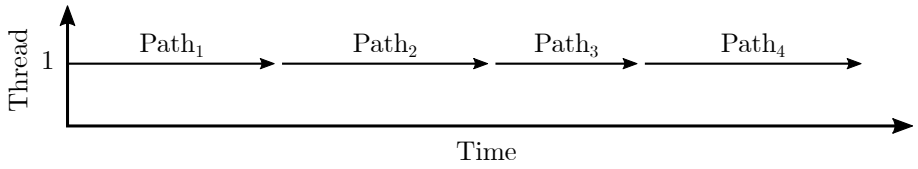


Figure 4.11: Planning paths sequentially for agents.

If inter-agent collisions are neglected, the searches can be completely decoupled. It is possible to plan the trajectories for the individual UAVs in parallel, as illustrated in Figure 4.12. Instead of planning each path one after the other, the paths are planned at the same time. However, it should be noted that there is some overhead to computing

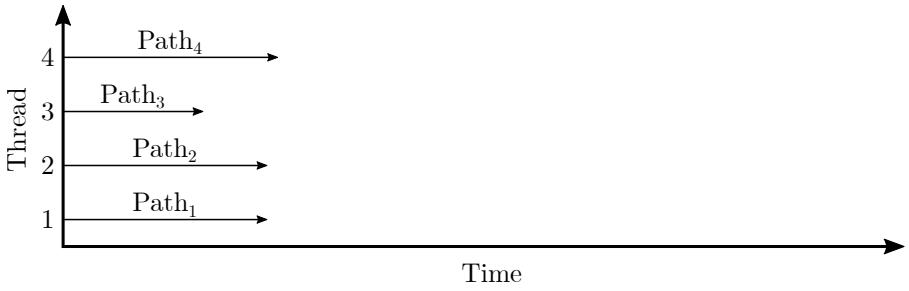


Figure 4.12: Planning paths in parallel over multiple threads.

the trajectories in parallel. Nevertheless, significant performance gains can be achieved if implemented correctly.

This project makes use of the A* Algorithm and its excellent performance, given a small search space and low-dimensional problem. However, it would also be possible to consider other search techniques, such as a sampling-based method.

4.6.1 Formulation

This section describes the formulation of the path planning problem to be solved by the A* algorithm. As described in Section 2.2.2, a grid-based search strategy discretises the state-space into a grid-like structure. This grid structure can be viewed as a graph. The discretised states corresponds to nodes within the graph. The inputs of the system is discretised to a set of actions. These actions corresponds to the edges of the graph structure. The following concepts are defined for a discrete path planning problem:

1. A non-empty *state-space* $\mathbf{X}$. The state-space represents all possible states of the system.

2. A finite *action space* $\mathbf{U}(\mathbf{x})$. The action space represents the actions, corresponding to discretised inputs $\mathbf{u}$, that the planner can apply to reach the goal position.

3. A *state transition function* $\boldsymbol{f}(\mathbf{x}, \mathbf{u})$. This function describes how a new node, $\mathbf{x}'$, can be generated given a current node, corresponding to the state $\mathbf{x}$, and an action, corresponding to a input $\mathbf{u}$.

4. An *initial state* $\mathbf{x}_i \in \mathbf{X}$.

5. A *goal set* $\mathbf{X}_G \in \mathbf{X}$.

State-space

The state-space should sufficiently describe the system. The state is selected as

$$\mathbf{x} = [x, y, z]^\intercal \in \mathcal{R}^3 \tag{4.13}$$

where x, y and z describes the positions in the world reference frame. The state space is discretised into a grid-like structure. Obstacles are excluded from the state-space by marking elements with obstacles as occupied.

Finite action space

The *action space* $\mathbf{U}(\mathbf{x})$ contains the set of actions $\mathbf{u}$ that the search algorithm can apply at the current node. The action space consists of actions moving along the grid axis, as well as diagonal actions. A vector represents each action. For example, the action to move along the $\mathbf{x}$ axis is defined as $\mathbf{u} = [1, 0, 0]^\intercal$

State transition function

The state transition function describes the effect of an action on a node. Formally the state transition function is defined as

$$\boldsymbol{f}(\mathbf{x}, \mathbf{u}) = \mathbf{x} + \mathbf{u} \tag{4.14}$$

where $\mathbf{x}$ is the current node to which the actions are applied, and $\mathbf{u}$ the input.

The initial state and goal set

The initial state is the starting position of the UAV at the current time-step. The goal state is the ideal position of the UAV at the end of the time horizon. However, it could be that the point falls within the boundaries of an obstacle, and thereby be impossible to reach. Therefore, it is necessary to define a secondary condition to terminate the search. A maximum number of search nodes are defined to ensure that the search is terminated. If the goal state is not reached, the node closest to the ideal position is selected as the goal node. The maximum number of search nodes is selected to be large enough to explore the surrounding area, yet small enough to not hold up the planning process.

The objective function and heuristic

The objective function was selected to obtain the shortest path to the goal position. Therefore the cost function is defined as the distance that was travelled.

The A* algorithm relies on an estimate of the cost to reach the goal set (heuristic). The heuristic is selected as the Euclidean distance to the goal location. Formally it can be expressed as

$$\text{cost\_to\_go} = \sqrt{(x_g - x)^2 + (y_g - y)^2 + (z_g - z)^2} \tag{4.15}$$

where the subscript g denotes the goal location. For the A* algorithm to be an optimal search algorithm, the heuristic should always underestimate the cost to reach the goal. This is true for the Euclidean heuristic. If the heuristic is selected to be greedy, it could decrease the search time, but the optimality guarantee is lost.

4.6.2 A* Algorithm

Initially created by Hart *et al.* [36], the A* Algorithm is a search algorithm to find the optimal path through a graph structure. The algorithm follows the same template as a general discrete planning algorithm, with the expansion of an estimate (heuristic) to reach the goal area. The background information in this section is adapted from the textbook on planning algorithms written by LaValle [8]. Algorithm 1 gives the pseudo-code for a general forward search algorithm. The basic operation of the algorithm is:

The algorithm takes a node, $\mathbf{x}$, from a priority queue, Q, and applies the set of actions $\mathbf{U}(\mathbf{x})$ through the state transition function $\boldsymbol{f}(\mathbf{x}, \mathbf{u})$. By applying the actions to the current node (parent node), new nodes are generated (child nodes). These nodes are added to the priority queue to be investigated later. This process is repeated until the node that is investigated falls within the goal set. The order in which the priority queue is sorted depends on the algorithm that is used. With the A* algorithm, the queue is sorted by the cost to reach the current node (*cost to come*), plus an estimate to reach the goal state (*cost to go*).

Algorithm 1 Template for forward search algorithm [8]

```
1: Q.Insert(x_i) and mark x_i as visited          ▷ Insert initial state into the queue
2: while Q not empty do
3:     x ← Q.GetFirst()                           ▷ Get first state in the queue
4:     if x ∈ X_G then
5:         return SUCCESS          ▷ Return success if state matches the goal state
6:     for u ∈ U do
7:         x' ← f(x, u)            ▷ Calculate next state for all inputs u in U
8:         if x' not visited and within bounds then
9:             Mark x' as visited
10:            Q.Insert(x')                       ▷ Add new state in the queue
11:        else
12:            Resolve duplicate x'
13: return FAILURE                                ▷ X_G not reached
```

4.7 Interpolation

The optimisation algorithm returns a set of discrete points in the form of decision variables. To obtain the trajectories from the decision variables, the discrete points needs to be interpolated. Every collocation technique has a corresponding interpolation technique. For trapezoidal collocation, the selected collocation method, the input can be interpolated with linear interpolation, and the state can be interpolated by a quadratic spline [60].

Trapezoidal collocation assumes that the control varies linearly between knot points. Therefore, linear interpolation can be used to obtain control values (acceleration commands). The control for t in a segment $t \in [t_k, t_k + 1]$ is expressed as

$$\mathbf{u}(t) \approx \mathbf{u}_k + \frac{t - t_k}{t_s}(\mathbf{u}_{k+1} - \mathbf{u}_k) \tag{4.16}$$

For trapezoidal collocation, the state trajectory is commonly represented by a quadratic spline [44]. The dynamics are approximated over a single segment $t \in [t_k, t_k + 1]$ as

$$\boldsymbol{f}(t) = \dot{\mathbf{x}}(t) \approx \boldsymbol{f}_k + \frac{(t - t_k)}{t_s}(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k). \tag{4.17}$$

However, the goal is to obtain $\mathbf{x}$, and not $\dot{\mathbf{x}}$. To obtain $\mathbf{x}$, both sides of the state equations can be integrated with the initial condition $\mathbf{x}(0) = \mathbf{x}_k$, yielding

$$\mathbf{x}(t_k) = \int \dot{\mathbf{x}}\, \mathrm{d}\tau \approx \mathbf{x}_k + \boldsymbol{f}_k t + \frac{(t - t_k)^2}{2(t_k - t_{k+1})}(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k). \tag{4.18}$$

4.8 Desired yaw angle

To describe the full state trajectory of a UAV, the trajectory must include both the position trajectory of the UAV's centre of mass and its yaw angle trajectory. As stated earlier, the yaw of each UAV does not affect the collisions. For this reason, the design choice has been made to plan the yaw angle after the collision-free position trajectories are planned.

The desired yaw angle of the vehicle depends on how the camera system is mounted on the UAV. Two strategies for mounting a camera on a UAV are illustrated in Figure 4.13. With a forward-facing camera, the camera is always facing in the direction that the UAV is pointing. However, if the camera is fitted to a rotating gimbal, the camera angle is not dependent on the UAV yaw angle.

Forward-facing camera

Gimbal-mounted camera

Figure 4.13: Strategies for mounting a camera on a UAV [77]. A fixed forward-facing camera requires the UAV to face the subject that it is capturing. A gimbal-mounted camera can operate independently of the UAV heading.

If a forward-facing camera is used, the yaw angle of the UAV must be controlled so that the camera points towards the target. Figure 4.14 illustrates the desired yaw angle. It is possible to determine the desired yaw angle by calculating the inclination of the line between them. The desired yaw angle can be expressed as[1]

$$\begin{aligned} \psi_{\text{ref}} &= \arctan\left(\frac{\Delta y_W}{\Delta x_W}\right) \\ &= \arctan\left(\frac{y_{\text{target}} - y_{\text{UAV}}}{x_{\text{target}} - x_{\text{UAV}}}\right). \end{aligned} \tag{4.19}$$

4.9 Online target following

The discussions up to now have assumed that the trajectory planning is performed once for the fixed prediction horizon and that the UAVs then just follow their planned reference trajectories. This would be an effective strategy if the target follows its predicted trajectory and if an infinite prediction horizon was used. However, the UAVs will eventually reach the end of the planning window, or the target may deviate from its predicted trajectory. A replanning strategy is therefore introduced to continuously replan for a receding planning horizon into the future and to adapt to changes in the target's predicted trajectory.

[1]To avoid a singularity when $x_{\text{target}} - x_{\text{UAV}} = 0$, a 4-quadrant inverse tangent function, such as arctan2 available in Numpy Python [78], can be used.

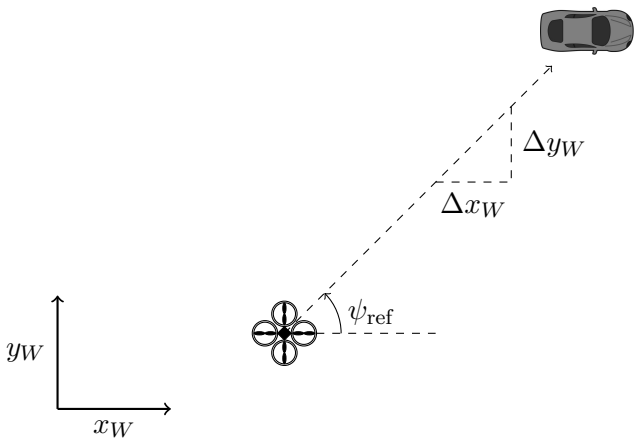


Figure 4.14: Definition of the desired yaw angle by calculating the inclination of the line between the target and the UAV.

4.9.1 Replanning strategy

The replanning strategy is similar to the MPC control strategy presented in Section 3.7. The concept is illustrated in Figure 4.15. At the first time-step, t_1, the trajectory of the target is predicted using its current position and velocity. This prediction is used to plan collision-free trajectories for the UAVs. The planned trajectory is published to the trajectory tracker. At the next time-step, t_2, the target has deviated from its predicted trajectory. The planner predicts the new trajectory of the target and replans new trajectories for the UAVs. This process is continuously repeated at a fixed replanning rate for the duration of the target following task.

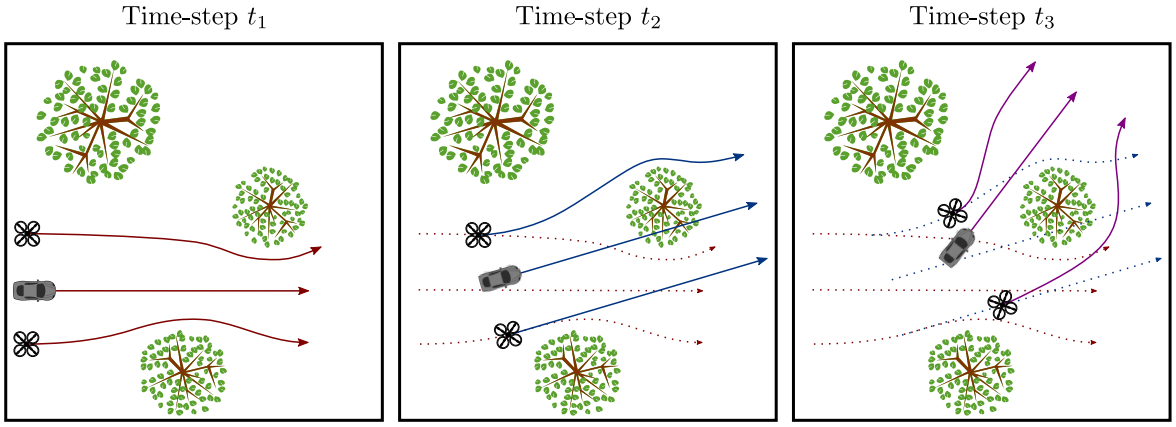


Figure 4.15: Replanning strategy to replan trajectories online as the target deviates from its predicted trajectory.

As illustrated in Figure 4.15, the UAVs do not execute their complete planned trajectories. The UAVs only execute the first part of their trajectories while planning new trajectories. The time horizon considered by the trajectory planner is generally much longer than the replanning interval. For this project, a typical time horizon of about 5 seconds was chosen with a replanning interval of about 250 ms. The trajectory planner would therefore plan the UAV trajectories for 5 seconds into the future, but the UAVs would only execute the first 250 ms while the trajectory planner already started planning for the next planning window. The impact of both the planning horizon length

and the replanning rate will be investigated in more detail in Chapter 6.

Figure 4.16 shows the timeline of the replanning strategy. The trajectory is replanned at a fixed replanning interval for time instants t_k, t_{k+1}, t_{k+2}, and so forth. The trajectory is planned for a planning horizon into the future, with the length of the planning horizon longer than the replanning interval. Assume that the trajectory planner has already planned Trajectory$_k$ to be executed starting from time instant t_k, and has planned the trajectory for the duration of the entire planning window, which extends beyond time instant t_{k+1}. The UAVs start executing their individual trajectories at time instant t_k, while the trajectory planner starts planning the next Trajectory$_{k+1}$ to be executed starting at time instant t_{k+1}. The trajectory planner must publish the new trajectory by the next time instant t_{k+1} which means that the planning algorithm must execute within a maximum processing time less than or equal to the replanning interval. Since the trajectory planner must start planning the next trajectory while the current trajectory is still being executed, it uses the planned future positions of the UAVs as the initial positions for the next trajectory starting at time-step t_{k+1}.

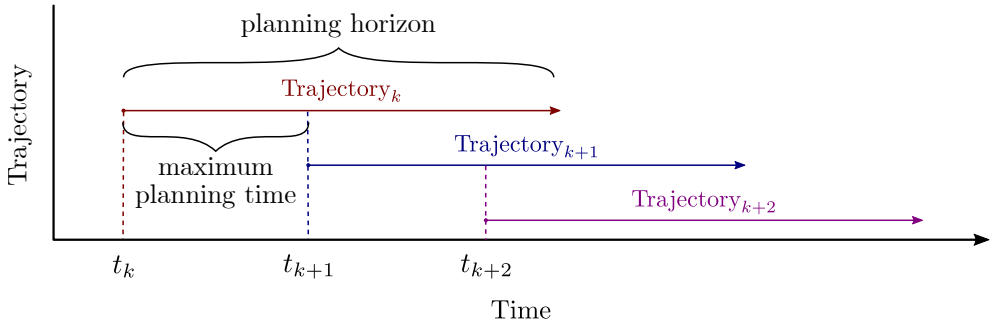


Figure 4.16: Trajectory planner timing diagram to show the planning window and replanning interval.

4.9.2 Recursive feasibility

Because the replanning strategy only plans up to a finite time horizon into the future, some way is needed to ensure that the planned trajectory at time-step t_k does not result in an infeasible trajectory at time-step t_{k+1}. This is known as ensuring *recursive feasibility* [68]. In control problems, this can be achieved by enforcing a constraint that the endpoint of the trajectory should be a stable fixed point. The concept can be expanded to trajectory planning. A potential stable point could be when all the UAVs are hovering, as this is a state in which no collisions occur.[2] The constraint is enforced at the end of the prediction horizon (which is approximately 5 seconds into the future). Therefore, when the trajectory optimisation strategy is used in the receding horizon configuration, the original objective function F is modified as shown in Equation 4.20 to ensure recursive feasibility. The objective for all UAVs to hover at the end of their trajectories is represented by a terminal cost in the objective function (a soft constraint) which heavily penalises non-zero UAV velocities at the final time instant by a large

[2]The assumption holds for static environments, but may not hold if future projects introduce dynamic obstacles.

weighting coefficient $w \approx 50$.

$$F_{\text{MPC}}(\boldsymbol{x}) = F(\boldsymbol{x}) + w \sum_{i=1}^{A} (\dot{\mathbf{p}}_i^{(N-1)})^2 \tag{4.20}$$

4.9.3 Handling planning errors

A disadvantage of the trajectory optimisation approach is that it is quite challenging to guarantee that the optimisation algorithm will execute and provide a solution within a given allowed planning time. The execution time of the trajectory optimisation algorithm should preferably be consistent and shorter than the replanning interval. However, the algorithm may sometimes take longer to execute if the optimisation problem is particularly challenging due to the specific target trajectory, the UAV initial states, or the layout of the environment. Contingency measures should therefore be implemented to handle the case where the trajectory planner does not provide a new planned trajectory in time.

An advantage of the constraint ensuring recursive feasibility (Section 4.9.2) is that it ensures that the UAVs will come to a safe standstill at the end of their planned trajectories. If the trajectory planner fails for a given planning interval, either due to an infeasible problem being posed or if the trajectory cannot be generated within the available time frame, the trajectory tracker will continue to follow the previously planned trajectory. The replanning strategy will then skip the failed planning iteration and will replan for the next planning window that starts at time instant t_{k+2}. The replanning is therefore continually repeated until a planned trajectory is calculated in time for the next time instant from which the trajectory execution starts. Should the trajectory planner completely fail for some reason, the UAVs will execute their last planned trajectories which end in safe hovering positions.

4.10 Trajectory planning example

This section presents an example of how the trajectory planner generates the trajectories with trajectory optimisation to verify that the approach works as expected. Figure 4.17 shows the example scenario that is provided to the trajectory planner. Two UAVs are tasked to follow a target vehicle which is travelling through a gap between two static obstacles in the environment. The two UAVs are instructed to maintain their relative positions on either side of the target.

The trajectory planning problem is formulated as an optimal control problem. The grid-search strategy is used to provide an initial estimate for the solution of the resulting NLP. The NLP is solved with the IPOPT algorithm. The optimisation algorithm successfully determines a solution that minimises the objective function, satisfies the dynamic constraints of the UAVs, and avoids inter-UAV collisions and collisions with the static environment after 19 iterations.

The state of the decision variables was stored after every optimisation iteration. This proved to be a powerful strategy to verify that the constrained optimisation algorithm was converging on the solution as expected. Table 4.1 shows the trajectories of the two UAVs at a few selected iterations to see how the trajectory progresses through the iterative optimisation. The table starts with the initial trajectories provided by the grid-based search algorithm and ends with the final trajectories after 19 optimisation

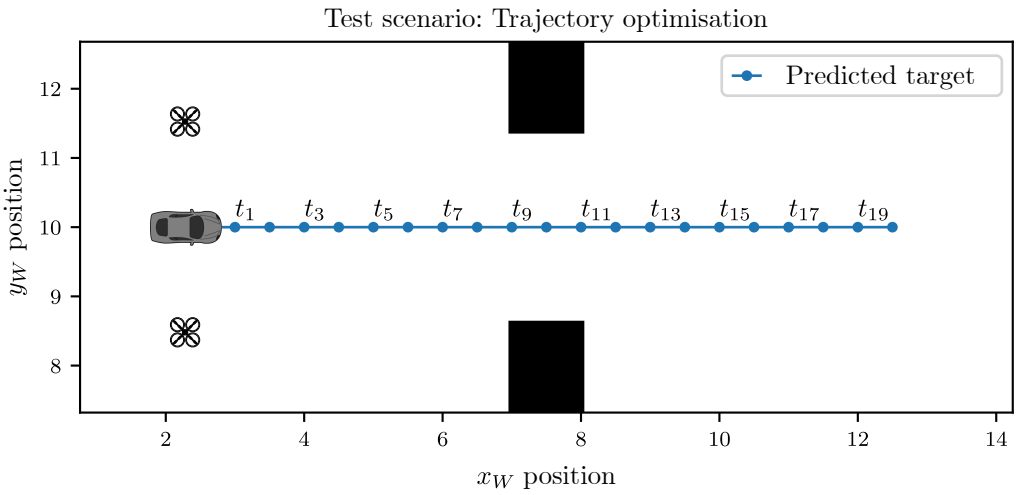


Figure 4.17: Example scenario to illustrate the operation of the trajectory planner.

iterations. At each iteration, the planned UAV trajectories are visualised, the values of the objective function and its components (target following and energy usage) are given, and the satisfaction of the dynamic constraints and the collision constraints are checked.

Table 4.1: Path at different iterations

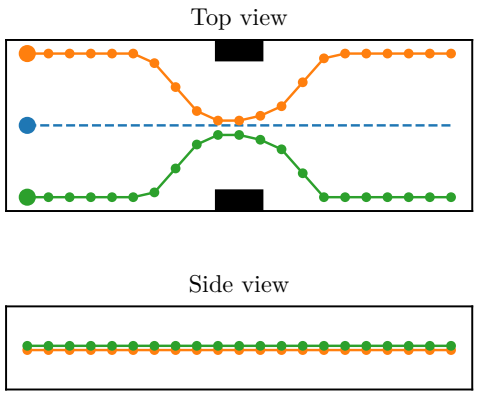
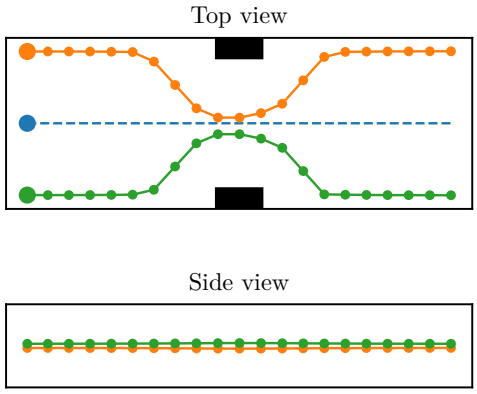
Path	Description
	Iteration 0 Objective: Energy 19.2 Objective: Target 16.81 Objective: Total 36.01 Nearest UAV 0.316 m (✗) Nearest Obstacle 1.181 m (✗) Dynamics Error 2.0 (✗) Max velocity 1.414 $\mathrm{m\,s^{-1}}$ (✓) Max acceleration 2.0 $\mathrm{m\,s^{-2}}$(✓)
At iteration 0, the optimisation has not started. The grid-based search algorithm has provided the initial values for the decision variables. The solution almost satisfies the obstacle constraints (the distance to the nearest obstacle must be at least 1.2 m). However, the inter-UAV collision constraints are not satisfied (the minimum distance between the UAVs must also be at least 1.2 meters), as the grid-search does not consider inter-UAV collisions. The *energy objective* corresponds to acceleration regularisation within the objective function, and the *target objective* corresponds to the total position error (Equation 4.6). The *dynamics error* corresponds to the error within the function approximation of the collocation method.	

Table 4.1 – Continued on next page

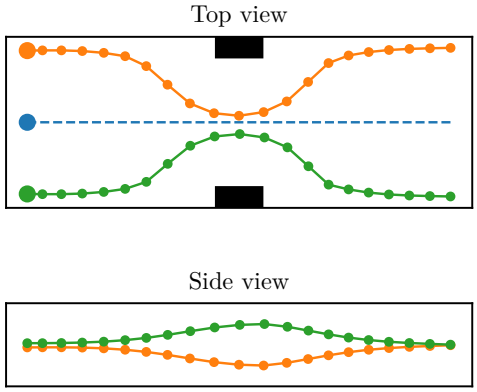
Table 4.1 – Continued from previous page

Path	Description



Iteration 1	
Objective: Energy	17.936
Objective: Target	16.385
Objective: Total	34.294
Nearest UAV	0.37 m (✗)
Nearest Obstacle	1.181 m (✗)
Dynamics Error	1.8829 (✗)
Max velocity	1.399 $\mathrm{m\,s^{-1}}$ (✓)
Max acceleration	1.918 $\mathrm{m\,s^{-2}}$ (✓)

After the first iteration, some progress is made with regard to the objective function and collision constraints. Visually, the path has not changed much. This shows that even a small positional change can improve the optimality of the solution. The enforced constraints are not yet satisfied.



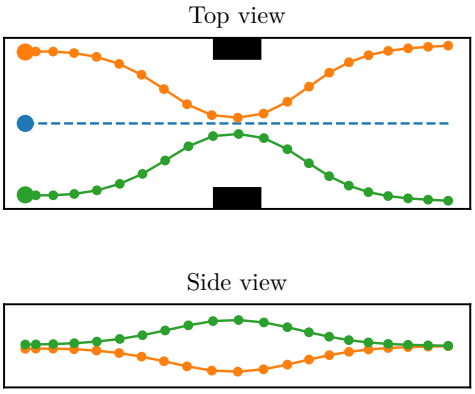
Iteration 3	
Objective: Energy	9.89
Objective: Target	18.46
Objective: Total	28.35
Nearest UAV	0.9522 m (✗)
Nearest Obstacle	1.195 m (✗)
Dynamics Error	0.84899 (✗)
Max velocity	1.326 $\mathrm{m\,s^{-1}}$ (✓)
Max acceleration	1.139 $\mathrm{m\,s^{-2}}$ (✓)

After three iterations, the paths are visually significantly smoother. This corresponds with the decrease in the energy cost of the trajectory, which means that the UAVs are accelerating less aggressively. The target-following cost has slightly increased, but the energy cost has significantly decreased, resulting in a net decrease in the total trajectory cost. Significant progress has also been made towards the satisfaction of the inter-UAV collision constraints. The minimum inter-UAV separation of 1.2 m has not been achieved yet but has significantly increased from 0.37 m to 0.95 m. This is due to the UAVs separating in height, shown in the side view.

Table 4.1 – Continued on next page

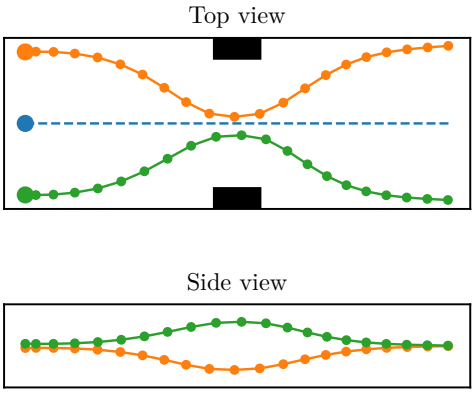
Table 4.1 – Continued from previous page

Path	Description



Iteration 4	
Objective: Energy	7.77
Objective: Target	21.79
Objective: Total	29.56
Nearest UAV	1.219 m (✓)
Nearest Obstacle	1.210 m (✓)
Dynamics Error	5.255e-14 (✓)
Max velocity	1.283 $\mathrm{m\,s^{-1}}$ (✓)
Max acceleration	0.999 $\mathrm{m\,s^{-2}}$ (✓)

From iteration three to iteration four, the total trajectory cost increases. However, all of the collision constraints and the dynamic constraints are now satisfied. The inter-UAV separation distance and the distance from each UAV to their nearest obstacle are now both greater than 1.2 m, and the error in the approximation of the dynamics is now within the acceptable tolerance. This means that the UAV trajectories are now collision-free and dynamically feasible.



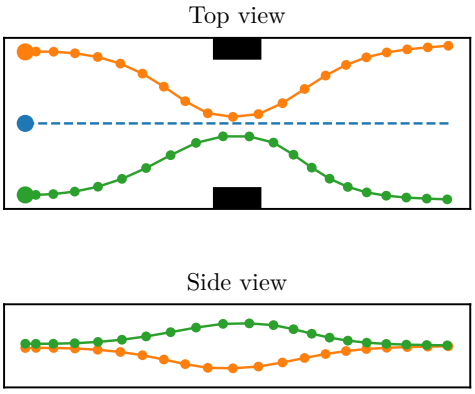
Iteration 8	
Objective: Energy	7.388
Objective: Target	20.956
Objective: Total	28.344
Nearest UAV	1.2009 m (✓)
Nearest Obstacle	1.2002 m (✓)
Dynamics Error	4.558e-14 (✓)
Max velocity	1.23 $\mathrm{m\,s^{-1}}$ (✓)
Max acceleration	1.009 $\mathrm{m\,s^{-2}}$ (✓)

After eight iterations, all of the constraints are still satisfied. The UAVs avoid each other as well as the static obstacles in the environment. Since the previously shown iteration, the target-following cost has increased, but the total acceleration needed to execute the trajectories cost has decreased.

Table 4.1 – Continued on next page

Table 4.1 – Continued from previous page

Path	Description



Iteration 19	
Objective: Energy	7.432
Objective: Target	20.778
Objective: Total	28.211
Nearest UAV	1.200001 m (✓)
Nearest Obstacle	1.200001 m (✓)
Dynamics Error	4.7703e-14 (✓)
Max velocity	1.323 $\mathrm{m\,s^{-1}}$ (✓)
Max acceleration	1.013 $\mathrm{m\,s^{-2}}$ (✓)

After 19 iterations, the optimisation algorithm has converged to a solution. All of the constraints are satisfied within the allowed tolerance. Since iteration eight, the solution has not changed significantly. Most of the iterations only improve the numerical precision of the solution but do not practically improve the planned trajectories.

Table 4.1 – End

Initially, rapid progress is made concerning both the objective function and the satisfaction of the constraints. However, as the optimisation algorithm nears the final solution, the progress slows and the improvement per iteration becomes smaller. In this example, the algorithm was practically converged after eight iterations but continued to make marginal progress until the termination criteria were reached in iteration 19. This indicates that less stringent termination criteria could possibly be used. However, investigating this aspect is left for future research projects. Another interesting behaviour to take note of is that the transcription method tends to improve the solution on all fronts in each iteration. At each iteration, progress is made in terms of both the objective function and the satisfaction of the constraints. Further discussion of features of the optimisation process and how it progresses is presented in Appendix A.

4.11 Summary

This chapter presented the design of the trajectory planner. This included the trajectory optimisation approach, a grid-search strategy for the initial estimate of the solution, and the replanning strategy. Finally, an example scenario was used to illustrate the execution of the trajectory planner and how it iteratively optimises the planned trajectories for the UAVs. The example showed that the trajectory planner is capable of generating trajectories that optimise the objective function and satisfy the constraints. The example also provided some insight into how the optimisation algorithm solves optimal control problems.

Chapter 5

System Implementation

This chapter describes the implementation of the complete target-following system and the simulation environment. First, the high-level system architecture is presented, and then the implementation of each of its components is discussed. The detailed design of the trajectory planner has already been described in the previous chapter, so this chapter focuses on the implementation of the remaining components, namely the trajectory tracker, the attitude controller, the rotary-wing UAVs, and the communication system. The components of the target-following system were implemented as nodes in the Robot Operating System (ROS) environment. The chapter concludes with a description of the simulation environment that was used to test the target-following system. The simulation environment was created in Gazebo and includes simulation models of the UAVs, the ground target, and the physical environment.

5.1 Architecture

The architecture of the complete target-following system is shown in Figure 5.1. The components of the system include the *target trajectory prediction*, the *environment model*, the *trajectory planner*, and the *trajectory execution*. The *target trajectory prediction* estimates the current state of the target and extrapolates its future trajectory. The *environment model* consists of a map of the environment and the ESDF generator. The *trajectory planner* consists of the grid-search algorithm to provide the initial solution, and the trajectory optimisation to perform the cooperative trajectory planning. The *trajectory planner* plans the individual UAV trajectories using the target trajectory provided by the target prediction, and the ESDF provided by the environment model. The planned trajectories are then provided to the *trajectory execution* components of the individual UAVs. The *trajectory execution* component for each UAV consists of a *trajectory tracker*, an *attitude controller*, and a physical *rotary-wing UAV*. The *trajectory tracker* provides attitude references to the *attitude controller*, and the *attitude controller* actuates the rotors of the physical UAV. The communication between the system components is facilitated by the ROS environment.

5.2 Trajectory execution

As illustrated in Figure 5.2, trajectory execution consists of a trajectory tracker, an attitude controller and physical rotary-wing UAV. The trajectory tracker receives

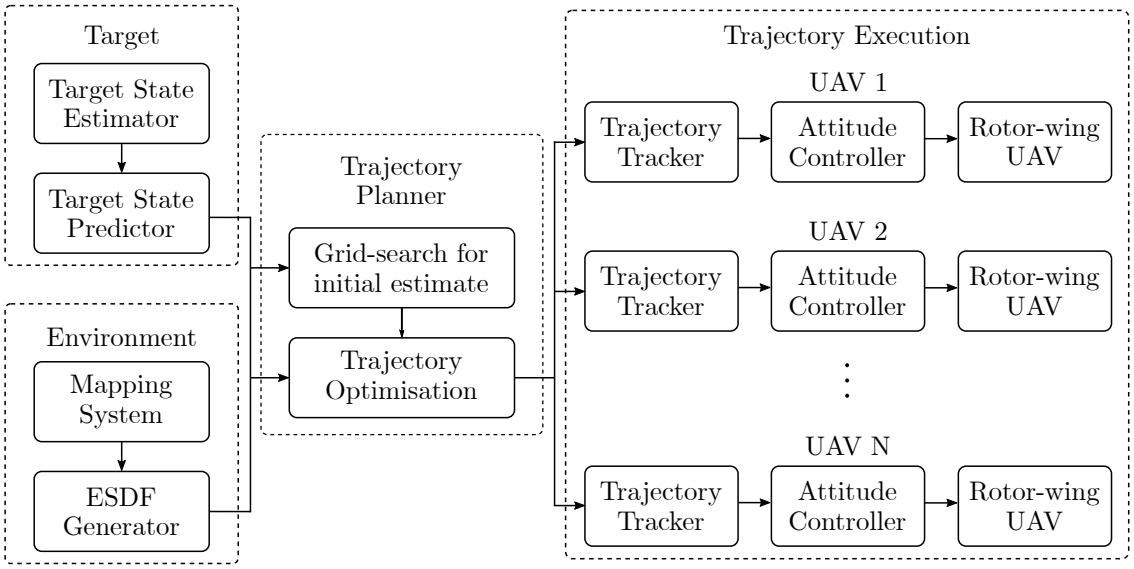


Figure 5.1: Architecture of the target-following system, with additional detail.

the reference trajectory $[\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_N]$ from the trajectory planner as a sequence of reference positions and reference yaw angles $\boldsymbol{\sigma} = [x, y, z, \psi]^\intercal$ for each time instant up to the planning horizon. The trajectory tracker controls the UAV to follow the position and yaw references by actuating the pitch angle θ_{ref}, roll angle ϕ_{ref}, yaw rate $\dot{\psi}_{\text{ref}}$, and total thrust T references for the UAV, which it feeds to the attitude controller. The attitude controller controls the UAV to follow the commanded pitch angle, roll angle, yaw rate and thrust references by actuating the individual rotor thrusts $\boldsymbol{T}_A$.

The attitude controller uses the estimated attitude and angular rates of the UAV as feedback. The trajectory tracker uses the full estimated state as feedback, which includes the position, linear velocity, attitude, and angular rates. An existing trajectory tracker and flight controller designed by Kamel *et al.* [11] were selected for the target-following system.

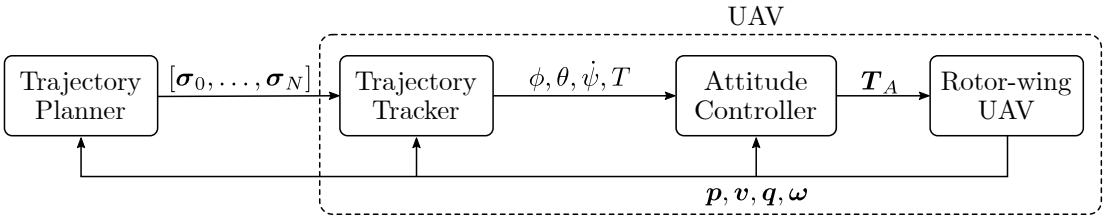


Figure 5.2: The architecture of a UAV which consists of a trajectory tracker, a flight controller and physical hardware.

5.2.1 Rotary-wing UAV

The AscTec Firefly hexarotor, shown in Figure 5.3, was chosen as a representative rotary-wing UAV for the implementation of the target-following system. The Firefly has a mass of 1.6 kg, a maximum speed of 50 km/h, and a payload-carrying capacity of 600 grams [79], making it suitable for target-following and carrying camera equipment. The flight controller and trajectory tracker chosen for the implementation of the target-following system have also been extensively tested on the AscTec Firefly [10], [11].

Figure 5.3: AscTec Firefly hexarotor [9].

5.2.2 Attitude controller

The attitude controller controls the attitude and the total thrust of the UAV. An existing attitude controller designed by Kamel *et al.* [11] was chosen for the implementation of the target-following system. The attitude controller was designed in conjunction with the trajectory tracker that will be discussed in the next section.

The architecture of the attitude controller is shown in Figure 5.4. The attitude controller controls the total thrust T, and the pitch angle, roll angle, and yaw rate of the UAV by actuating the individual rotor thrusts. The Firefly hexarotor UAV has six rotors that all point in the same direction. A mixer is used to translate the commanded total thrust T and the commanded body-axis moments τ_ϕ, τ_θ, and τ_ψ to six individual rotor thrust command $\boldsymbol{T}_A$. Two separate PID controllers are used to control the pitch angle and the roll angle, respectively. A third PD controller is used to control the yaw rate. The PID controllers use the estimated attitude and angular rates of the UAV to actuate the body-axis moments applied to the UAV.

By commanding the total thrust and the attitude of the UAV, the magnitude and direction of its thrust vector can be actuated to control the translational motion of the UAV in 3D space.

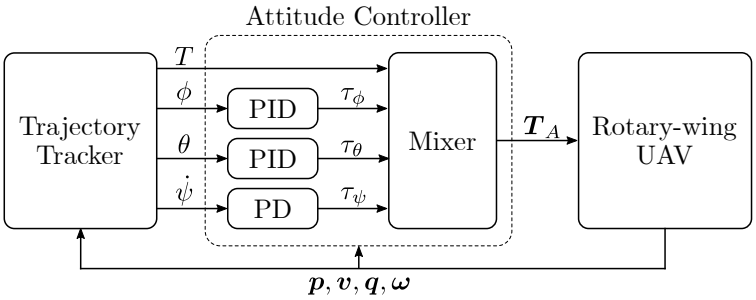


Figure 5.4: Architecture of the attitude controller.

5.2.3 Trajectory tracker

The trajectory tracker controls the translational motion and the yaw angle of the UAV to follow the reference trajectory provided by the trajectory planner. An existing trajectory tracker designed by Kamel *et al.* [11] was used for the implementation of the target-following system.

The trajectory tracker controls the translational and rotational motion of the UAV by actuating the thrust, pitch angle, roll angle, and yaw rate of the UAV by providing references to the attitude controller. The trajectory tracker uses a linear model predictive control architecture. The UAV model is linearised about the hovering condition and

the following optimal control problem is formulated and repeatedly solved

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u}} \quad & \sum_{k=0}^{N-1} (||\mathbf{x}_k - \mathbf{x}_{\text{ref},k}||^2_{\mathbf{Q}} + ||\mathbf{u}_k - \mathbf{u}_{\text{ref},k}||^2_{\mathbf{R}} + ||\mathbf{u}_k - \mathbf{u}_{k-1}||^2_{\mathbf{P}}) \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = \boldsymbol{A}\mathbf{x}_k + \boldsymbol{B}\mathbf{u}_k \\ & \mathbf{u}_k \in U \\ & \mathbf{x}_0 = \mathbf{x}(t_0) \end{aligned} \tag{5.1}$$

where $\mathbf{x}_{\text{ref}}$ and $\mathbf{u}_{\text{ref}}$ are the reference state trajectory and the reference control signal, $\mathbf{x}$ and $\mathbf{u}$ are the actual state and the actual control signal, and $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{P}$ are penalties on the state error, control error, and control change rate, respectively. The resulting optimisation problem is quadratic with linear constraints and is called a Quadratic Program (QP) problem. The resulting QP problem is then solved with a quadratic programming solver such as QPOASES [80]. The actual implementation of the trajectory tracker also includes additional components such as a disturbance estimator.

Interestingly, although the trajectory planner and the trajectory tracker perform different roles, their internal architecture is quite similar. Both components use trajectory optimisation and MPC. Both solve an optimal control problem at each iteration for a receding horizon. However, the role of the planner is to plan collision-free trajectories following the target, while the trajectory tracker is to receive the planned trajectories and execute them.

If the formulation of the trajectory tracker is expanded to avoid collisions with other UAVs and the environment, the result is an optimisation problem comparable with that of the trajectory planner. However, as seen in Chapter 4, this results in a highly non-linear problem which is more difficult to pose and solve. The more straightforward QP problem can be solved at a faster rate, making it suitable for a lower-level trajectory tracker. The similarities illustrate how the line between planning and control can become blurred.

5.3 Communication

The communication between the different components of the target-following system is facilitated by the Robot Operating System (ROS). The components of the target-following system are implemented as ROS nodes and communicate with one another using ROS messages.

5.3.1 The ROS environment

A big shift in the robotics industry came through the advent of the Robot Operating System (ROS). The ROS website [81] defines ROS as "a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms." ROS aims to create a framework where researchers write modules that can interact with the code of others, allowing researchers to build on the research of others. ROS can handle the communication within a robotics system and is used in this project between the different components.

5.3.2 ROS concepts

ROS uses a publish-subscribe architecture, where different software components are implemented as *nodes*, and publish *messages* to *topics*. Figure 5.5 presents a basic example of how a position control system for a UAV could be implemented in ROS.

The *nodes* are indicated in ovals, such as a *remote* and a *path planner* to send position commands to the *controller*, as well as a *quadcopter* node to execute the motor commands and measure the state of the vehicle. Each of these nodes can publish a message to a topic. The topics are indicated by rectangles.

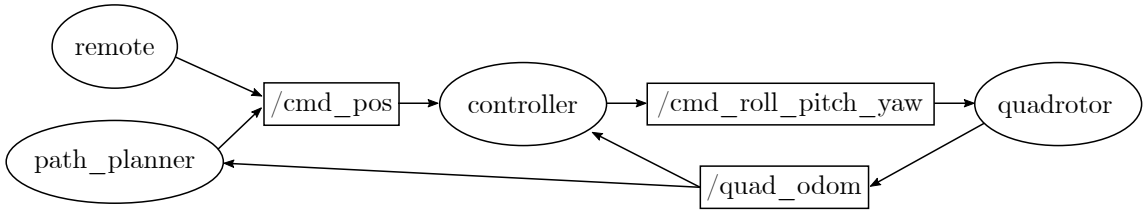


Figure 5.5: Example of a possible implementation for a feedback control system within a ROS architecture.

To publish to a topic, the message should have a predefined message type. Although custom message types are possible, a wide variety of standard messages exist within the architecture. These standard message types allow for modularity between various components and projects. ROS enables researchers to build on the work of others by providing an interface between components for researchers to use. This approach is leveraged in the target-following system, as it allowed for our trajectory planner to be integrated with an existing trajectory tracker and attitude controller designed by other researchers.

5.4 Simulation environment

A simulation environment was constructed to verify the performance of the trajectory planner. This section describes the construction of the simulation environment, which includes the maps of the environment, the targets that are being followed, and the representative UAV simulation model. The simulation model is implemented in a ROS environment, and is used in Chapter 6 and Chapter 7 to test the trajectory planner.

Figure 5.6 provides a visualisation of the simulation environment, in which four UAVs are following a target (arrow). The planned trajectories (for the planning horizon) are also shown (green lines). Video animations of a few scenarios are available here.[1]

5.4.1 Environment maps

The simulation environment makes use of randomly generated maps. Different categories of environments are defined to assess the versatility of the trajectory planner in various scenarios. Table 5.1 summarises the four types of categories, namely *deserts*, *savannas*, *forests* and *urban* environments. Each randomly generated map is 50m $\times$ 50m $\times$ 10m in size and has a resolution of 0.1pixels/m.

[1]URL to the video animations: `https://youtu.be/4sFs3dQOVWc`

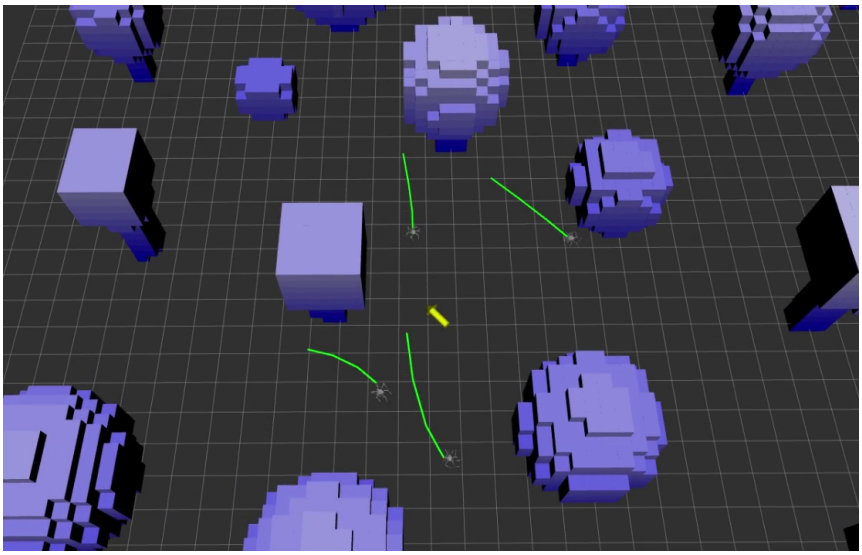


Figure 5.6: Four UAVs following a ground target through a cluttered environment.

Table 5.1: Categories of simulation environments

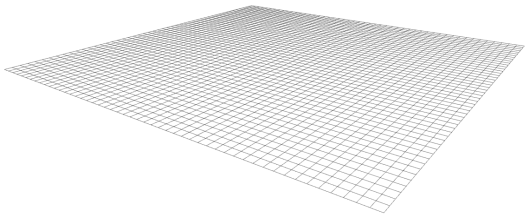
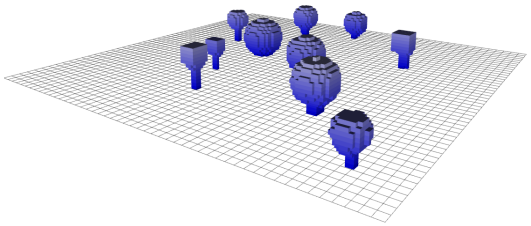
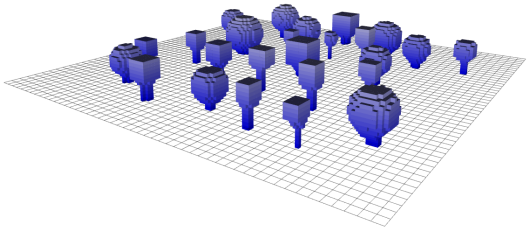
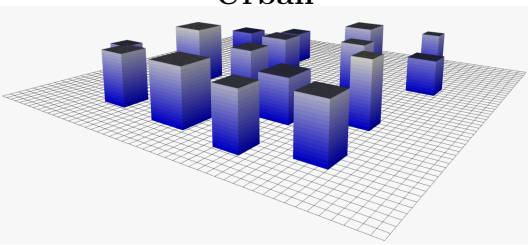
Environment	Description
Desert 	This simulation environment does not contain any obstacles. The scenario represents a flat desert environment. The map is introduced as a baseline to see how well the planner performs without any obstacles.
Savanna 	This simulation environment contains sparsely scattered trees. This represents a savanna scenario.
Forest 	This simulation environment is densely populated with trees. This represents a forest scenario.

Table 5.1 – Continued on next page

Table 5.1 – Continued from previous page

Environment	Description
Urban 	This simulation environment consists of randomly placed cuboids. This portrays an urban scenario where the cuboids represent buildings.

5.4.2 Ground target

The ground target is modelled as a point mass and its motion is simulated by applying random linear, and angular velocities to the following kinematic model

$$\dot{x}(t) = v(t)\cos(\psi(t)), \tag{5.2}$$
$$\dot{y}(t) = v(t)\sin(\psi(t)), \tag{5.3}$$

where v is the forward velocity and ψ the heading of the target, as defined in Section 4.3.1. For simplicity, it is assumed that the ground target can travel through obstacles in the environment.

By selecting the probabilistic distributions of the target's linear velocity v, and angular velocity $\dot{\psi}$, different types of target behaviour can be achieved. The target's linear velocity determines whether it is slow or fast, and its angular velocity determines whether it is sluggish or agile. Random targets were generated by sampling the linear velocity and angular velocities from a Gaussian distribution. The time between sampling the distributions were sampled from a uniform distribution.

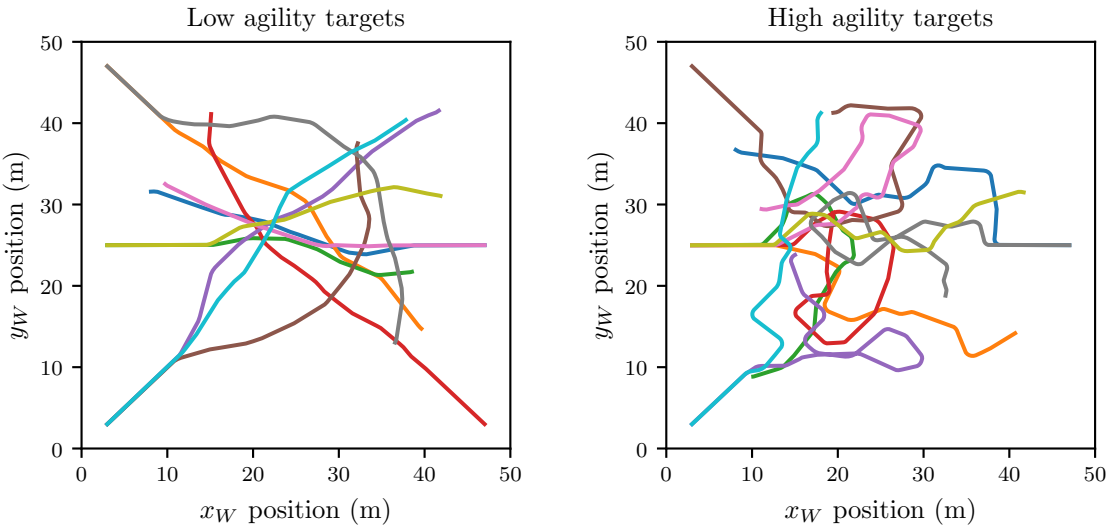
The mean and variance of each distribution is selected to correspond to the categories of targets, both in terms of agility and velocity. The two categories for target agility are defined as:

- **Low agility (0.3 rad/s):** Targets change direction gradually.
- **High agility (0.9 rad/s):** Targets change direction rapidly and frequently.

Figure 5.7 (a) and (b) show examples of randomly generated low agility and high agility targets, respectively. The three categories for average target velocity are defined as:

- **Slow (5 km/h):** Corresponds to following a person that is walking.
- **Medium (10 km/h):** Corresponds to following a person that is jogging.
- **Fast (15 km/h):** Corresponds to following a fast runner, or a casual cyclist.

This results in six types of targets that will be used to test the performance of the target-following system: (1) slow speed with low agility, (2) medium speed with low agility, (3) fast speed with low agility, (4) slow speed with high agility, (5) medium speed with high agility, and (6) fast speed with high agility.



(a) Low agility targets.

(b) High agility targets.

Figure 5.7: Randomly generated low agility and high agility targets within the simulation environment.

5.4.3 Rotorwing UAV simulation model

To verify if the trajectory tracker can execute the trajectories planned by the trajectory planner, a representative simulation model of the AscTec Firefly, developed by Furrer *et al.* [10], is used. This section describes the tools used to implement the simulation model, Gazebo and RotorS Simulator, and details regarding the mathematical model.

Gazebo

Gazebo is a high-fidelity simulator originally developed by Koenig and Howard [18]. Gazebo is highly integrated with ROS and allows for the simulation of various vehicles. It is capable of simulating differential equations describing the motion of the vehicle, as well as various sensors.

RotorS Simulator

RotorS Simulator, developed by Furrer *et al.* [10], is a modular Micro Aerial Vehicle (MAV) simulator with various vehicles, state estimators and flight controllers. It is built on top of the Gazebo simulator to simulate the vehicle physics. The flight controllers and state estimators are implemented as ROS nodes. The simulation models are verified with comparisons of recorded flight data and are accepted to be an accurate representation of an actual flight vehicle [10].

UAV simulation model

This subsection gives a short overview of the UAV simulation model. A derivation of the mathematical model is presented in Appendix B, which has been summarised from Furrer *et al.* [10].

Figure 5.8 presents a block diagram of the simulation components within the RotorS Simulator framework, which has been designed to closely mimic the structure of a physical UAV [10]. The dynamics of the FireFly UAV are simulated by the Gazebo physics engine. The Gazebo environment contains the simulated UAV dynamics, which consists of a rigid body with rotors fixed to the positions corresponding to the physical vehicle. Each rotor has motor dynamics which accounts for the most dominant aerodynamic effects [10]. To simulate realistic conditions, sensor noise is added to the dynamics to create the Inertial Measurement Unit (IMU) and pose measurements. RotorS also contain a state estimation component, based on an Extended Kalman Filter (EKF), for estimating the odometry from the IMU and pose measurements. The attitude controller, described in Section 5.2.2, makes use of the estimated odometry to control the UAV. A gazebo interface is written as a plugin to allow the attitude controller to apply desired rotor velocity commands to the simulated UAV model.

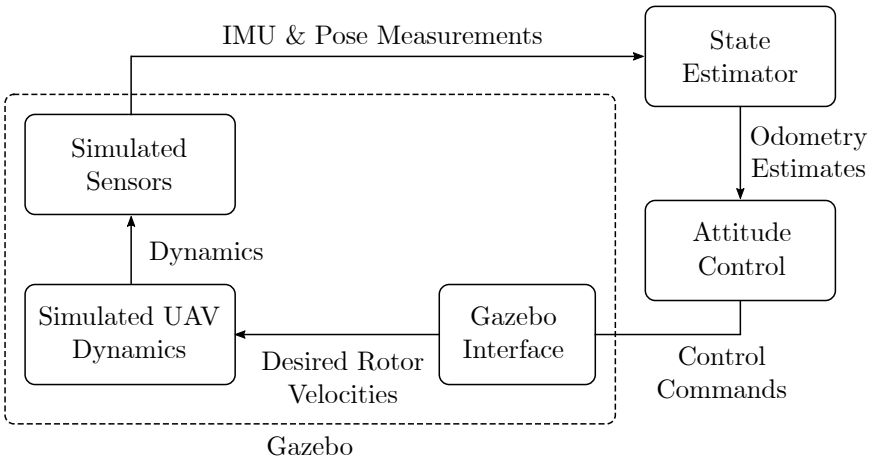


Figure 5.8: Simulation framework of the RotorS Simulator, adapted from Furrer *et al.* [10].

5.5 Summary

This chapter presented the architecture of the broader target-following system. It specifically focused on the interaction between the trajectory planner and the trajectory tracker. The chapter also described the creation of the simulation environment, which will be used in Chapter 6 and Chapter 7 to test the trajectory planner. The simulation environment allows different classes of environments containing random obstacles to be generated, and different types of targets to be simulated.

Chapter 6

Parameter Selection

Chapter 4 introduced many model parameters such as planning resolution, planning horizon, and replanning rate. The ideal trajectory planner would have an infinite planning horizon and would plan a very accurate, high-resolution path at a very fast replanning rate. However, a practical system has computational limits as to what it can achieve. Some of these parameters are conflicting (e.g. accuracy vs convergence) and limited to the available planning time. This chapter provides some guidance on how to select and design these parameters. The impact of the design parameters (planning resolution, planning horizon, and replanning rate) on the target-following performance are investigated both from a theoretical perspective and through simulation experiments.

6.1 Parameter investigation

This section investigates the design parameters from a theoretical perspective. First, the maximum acceleration and the velocity of a rotary-wing UAV is discussed, as it impacts the design parameters. This is followed by a theoretical discussion on the impact of the planning resolution, planning horizon length and replanning rate.

6.1.1 Vehicle velocity and acceleration

The selection of the design parameters depends heavily on the dynamic limitations (velocity and maximum acceleration) of the rotary-wing UAVs. We, therefore, start with a way to estimate the maximum horizontal acceleration of a UAV from its mass, maximum thrust, and maximum tilt angle.

The maximum (non-climbing) horizontal acceleration is calculated using the free body diagram of the rotary-wing UAV shown in Figure 6.1. The UAV has a tilt angle θ, a thrust T, and a mass m. The gravitational acceleration is g. The horizontal and vertical components of the thrust are denoted f_x and f_z respectively.

For the UAV to maintain level flight at its current altitude, the vertical component f_z of the thrust T must equal the gravitational force mg. Assuming that the UAV can produce sufficient thrust to counter the gravitational force, the horizontal component of the thrust can be expressed as

$$f_x = mg \tan(\theta) \tag{6.1}$$

and the maximum horizontal acceleration can be expressed as

$$a_x^{\max} = g \tan \theta^{\max}. \tag{6.2}$$

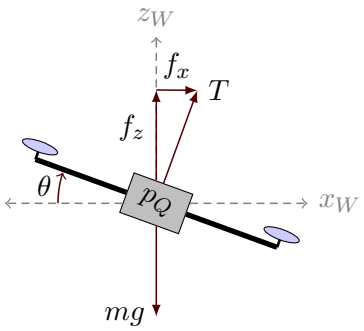


Figure 6.1: 2D Free body diagram of rotary-wing UAV.

The maximum horizontal acceleration provides a good indication of the agility of the UAV. However, the dynamic limitations of the rotary-wing UAVs are not only determined by the vehicle's capabilities, but also by the performance of the trajectory tracker that controls the UAV. The performance of the target-following system with the selected design parameters should therefore be verified with simulations that include the response of the trajectory tracker. The trajectory tracker is designed with a certain agility and velocity in mind. The design specifications of the trajectory tracker should not be exceeded. Furthermore, it is good practice to stay on the conservative side of the limits. This ensures that additional control energy is available when other disturbances, such as wind, are present.

6.1.2 Planning resolution

The *planning resolution* refers to the time resolution at which the collocation technique samples the trajectory. With trapezoidal collocation, the dynamics are accurate at the collocation points but are approximated between points [45]. The higher the planning resolution, the more accurately the planned trajectory will be represented. However, the trajectory tracker that executes the planned trajectory does not necessarily require a high-resolution representation of the trajectory, since it implements its own MPC controller with a higher accuracy model. It is therefore not essential that the trajectory planner provides a high-resolution reference trajectory, but rather that it provides a reference trajectory that is safe and dynamically feasible.

An advantage of a higher planning resolution is that it gives a more refined level of control over the trajectory, as illustrated in Figure 6.2. Having a more refined level of control allows the planner to plan better paths following the target. However, planning at a higher resolution increases the number of decision variables of the optimisation problem, increasing the computational complexity. There is, therefore, a trade-off between the path accuracy and the execution time.

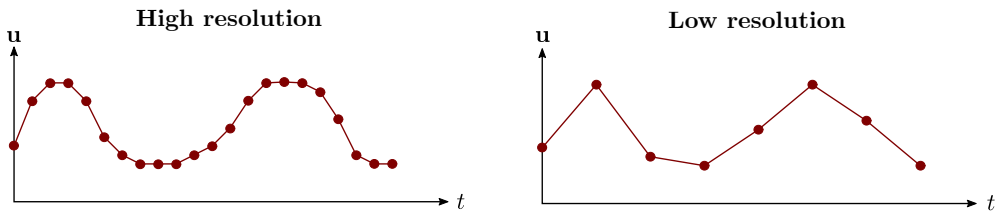


Figure 6.2: Effect of planning resolution on trajectory control.

The planning resolution also affects the planner's ability to plan safe trajectories around obstacles, as illustrated in Figure 6.3. The figure shows two paths, a high-resolution path as well as a low-resolution path around a point obstacle, $\mathbf{p}_{\text{obs}}$. With the high-resolution path, the trajectory goes around the obstacle. However, with the low-resolution path, a part of the trajectory goes through the safety radius around the obstacle. The obstacle constraints are only enforced at the collocation points and not in-between. Therefore, if the planning resolution is too low, the trajectory planner could plan trajectories that violate the distance constraint. To ensure that the trajectory of the vehicle satisfies the collision constraints, the maximum distance that the vehicle can travel between points should be less than the width of the safety radius. The distance d that a UAV can travel between collocation points is calculated as

$$d = vt_s, \tag{6.3}$$

where v is the UAV velocity and t_s is the planning resolution.

Figure 6.3: High-resolution vs low-resolution path around an obstacle point.

If the maximum distance between collocation points is limited to d_{min} (where d_{min} is the enforced safety radius), the worst case is illustrated in Figure 6.4. The trajectory will slightly violate the safety radius. To achieve this, the maximum time between collocation points t_s^{max} is calculated as

$$t_s^{\text{max}} < \frac{d_{\text{min}}}{v^{\text{max}}}, \tag{6.4}$$

where v^{max} is the maximum velocity of the UAV and d_{min} is the minimum allowable distance to an obstacle.

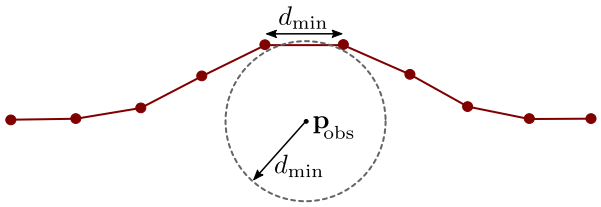


Figure 6.4: Worst case scenario if the maximum distance between collocation points is limited to d_{min}.

6.1.3 Planning horizon

The planning horizon t_{horizon} is how far into the future the trajectory planner generates trajectories based on the predicted target trajectory. The general idea is that the further the planner predicts into the future, the sooner it can anticipate future obstacles.

However, the further the planning horizon, the more computationally expensive it is to calculate the trajectory. Because the replanning strategy replans the trajectories at a fixed time-step, it may even be redundant to plan too far into the future. Another reason not to plan too far into the future is the uncertainty in the ground vehicle's future direction of movement.

To ensure the recursive feasibility (as described in Section 4.9.2), the trajectories are constrained so that the UAVs come to a standstill and hover at the end of their planned trajectories. As the trajectory planner must plan to bring the UAVs to a standstill by the end of the trajectory, the planned maximum velocity of the UAVs will automatically be limited (due to the limits imposed on the acceleration). On the other hand, the velocity of a UAV should at least match the velocity of the target it is following (otherwise it will fall behind). The planning horizon should also be distant enough so that the UAVs can accelerate and achieve the required matching velocity to follow the target. The planning horizon should therefore be at least long enough to allow the UAV to accelerate from its maximum speed to zero speed within the horizon, given the UAV's maximum acceleration capability. At a constant acceleration a_c, the change in velocity over a segment can be expressed as

$$v = v_0 + a_c t. \tag{6.5}$$

Rewriting the equation above, the minimum planning horizon $t_{\text{horizon}}^{\min}$ can be expressed as

$$t_{\text{horizon}}^{\min} > \frac{v}{a^{\max}}. \tag{6.6}$$

6.1.4 Replanning rate

The *replanning rate* is the rate at which the MPC strategy replans the trajectories. For the planner to react quickly to changes in the environment and target deviations, the replanning rate should be as high as possible. However, as discussed previously in Section 4.9.1, there is a limit to how fast the trajectory planner can replan trajectories. The replanning rate is limited by the time it takes to plan a trajectory. For a high replanning rate, a very fast and consistent trajectory planner is needed.

6.2 Parameter experiments

The previous section investigated the impact of the design parameters (planning resolution, planning horizon, and replanning rate) on the target-following performance from a theoretical perspective. In this section, the investigation is continued with simulation experiments.

A set of base parameters is defined, as summarised in Table 6.1. In each of the experiments below, all the parameters are kept constant except the parameter that is being investigated. In each of the experiments, the performance is evaluated in terms of *success rate*, *planning time* and the *target following objective*.

The experiments are conducted in the simulation environment described in Section 5.4.[1] Scenarios are created in the 'Savanna', 'Forest' and 'Urban' environments,

[1]The simulations were performed on a Lenovo Ideapad 330 with an Intel i7-8550U processor, 16Gb of RAM running Ubuntu 18.04. The grid search strategy was implemented in C++, and the trajectory optimisation was formulated in Python using Casadi [70].

Table 6.1: Base parameters for experiments

Item	Value
Number of agents	4
Planning resolution	0.3 s
Horizon length	4 s
Replanning rate	3.3 Hz
Maximum velocity	15 $\mathrm{m\,s^{-1}}$
Maximum acceleration	4 $\mathrm{m\,s^{-2}}$
Safety radius	1.2 m

with a combination of targets (slow and medium velocity, high and low agility). Each of the parameter combinations was tested on 21 target-following scenarios. This resulted in 43 677 trajectory-planning problems across all three experiments. These experiments were conducted to only evaluate the performance of the trajectory planner; this investigation did not include the trajectory tracker and its effects on the system.

6.2.1 Performance metrics

For each experiment, the performance of the trajectory planner is evaluated using the following metrics:

- **Success rate**: The success rate is the ratio of trajectory-planning problems solved to trajectory-planning problems attempted. It is defined in terms of each of the individual trajectory-planning problems within the replanning strategy. As explained in Section 4.20, a single failed iteration does not result in a failed scenario. It only means that an iteration of the replanning strategy was skipped, thereby degrading the target-following performance.
- **Planning time**: The planning time is the total time that the trajectory planner takes to compute the solution to a trajectory planning problem. The planning time includes the initial grid searches, the transcription process, and the execution of the NLP solver. To achieve real-time planning, the planning time should be shorter than the replanning interval.
- **Target-following error**: The target-following performance is measured as the average squared error between the desired position and the planned position at each time-step for the complete trajectory. The position error is calculated using Equation 4.6 (the objective function without acceleration penalty) that was defined previously.

6.2.2 Experiment 1: Planning resolution

This experiment investigates the impact of the planning resolution on the performance of the trajectory planner. The success rate of the trajectory planner as a function of the planning resolution is shown in Figure 6.5 (The success rate was determined for 14 067 planning problems). The graph shows both *successful* planning instances where the optimisation algorithm succeeded in obtaining a solution that satisfies the constraints, and instances that were successful and planned within the real-time limit imposed by

the replanning strategy. The graph shows that the algorithm maintains a high success rate throughout. However, the real-time planning ability degrades as the time between collocation points decreases (which results in a higher planning resolution).

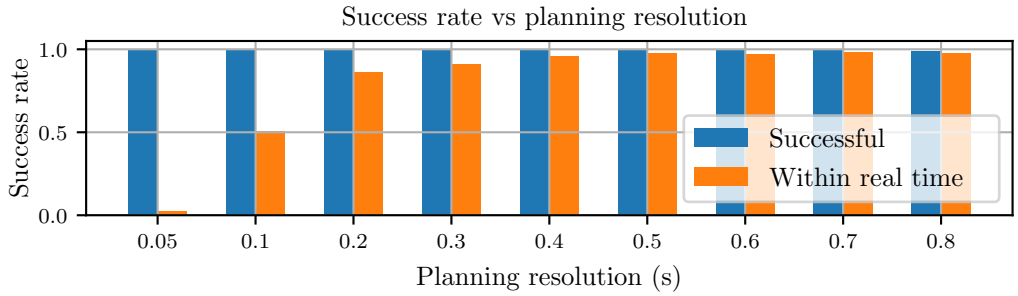


Figure 6.5: Success rate as a function of planning resolution.

Section 6.1.2 highlighted the risk that at a low planning resolution, the actual trajectory might violate the obstacle constraints. To measure the impact of this, Figure 6.6 presents the distance to the nearest obstacle (calculated with the ESDF) of the interpolated path. The box corresponds to the average distance and the whiskers to the maximum and minimum distances. The red line indicates the enforced safety radius. At low resolutions, the trajectory planner violates the actual minimum distance while still satisfying the optimisation constraints. This corresponds well to the theory presented earlier.

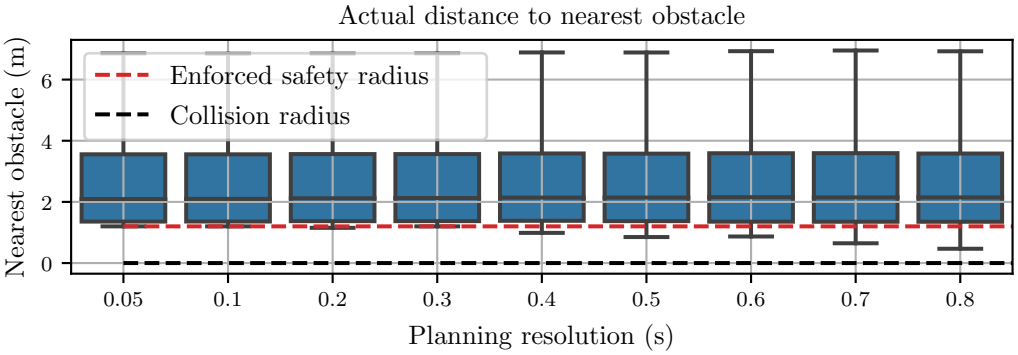


Figure 6.6: Actual minimum distance to nearest obstacle for each simulation.

Figure 6.7 shows the average planning time versus the planning resolution. The standard deviation of the average planning time is plotted as a confidence interval, as it shows how much the planning time varies. The graph shows that the planning time increases at higher resolutions. This corresponds to what was observed in Figure 6.5. A high-resolution path cannot be generated within the time frame allowed by the MPC strategy (with a sampling time smaller than 0.2 seconds for the current implementation).

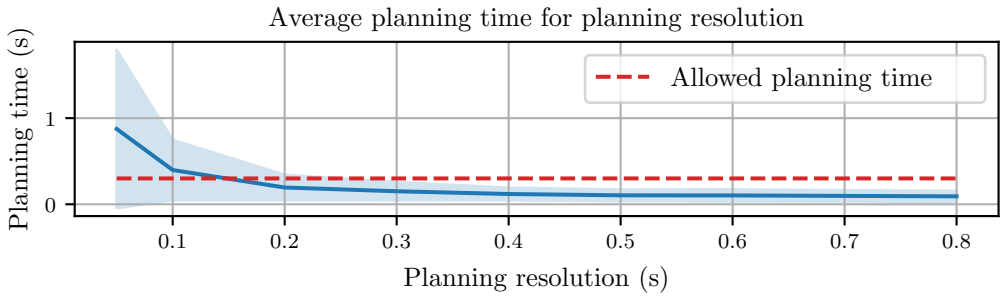


Figure 6.7: Planning time vs planning resolution.

Figure 6.8 shows the target-following error as a function of the planning resolution (as defined in Section 6.2.1). The shorter the time between collocation points, the better the target-following ability.

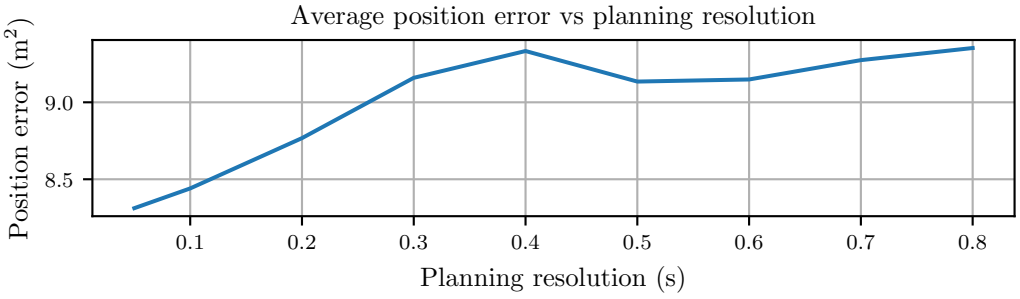


Figure 6.8: Objective function vs planning resolution.

This experiment confirms the theory that the higher the planning resolution, the higher the quality of the trajectory. However, this comes at the cost of increased computational burden. The experiment also shows that a low-resolution path might not adequately enforce a minimum distance from obstacles.

6.2.3 Experiment 2: Planning horizon

This experiment measures the impact of the planning horizon length on the trajectory planner. Figure 6.9 shows the impact of the horizon length on the *success rate* (The success rate was determined for 14 067 planning problems). As before, a distinction is made between successful results and results planned within the time limit. When the horizon length is extremely short, the success rate of the planner suffers, because the UAVs cannot keep up with the target. When the horizon length is long, the success rate increases. However, the longer the horizon length, the more the real-time planning capabilities of the planner suffer. This is confirmed by Figure 6.10, which shows the average planning time (with standard deviation). For long planning horizons, the planning time takes longer than the allotted planning time. As explained in Section 6.1.3, the size of the planning problem increases as the horizon length becomes longer.

Figure 6.11 presents the target-following deviation for different horizon lengths. As stated above, for a short horizon, the maximum velocity that the UAV can achieve is limited, which results in a very high position error. However, there is no significant

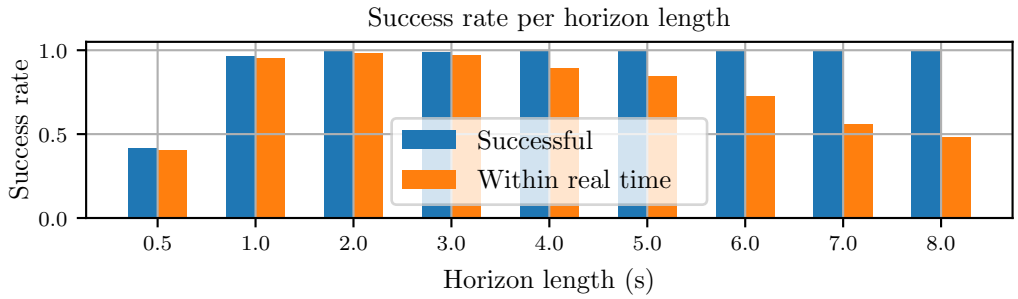


Figure 6.9: Success rate vs horizon length.

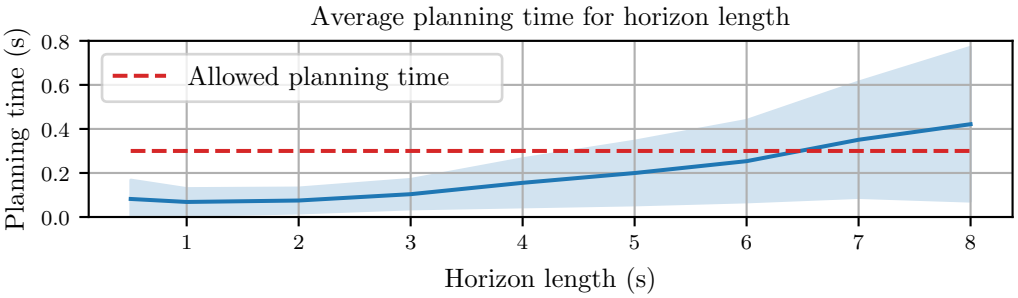


Figure 6.10: Planning time vs horizon length.

difference in target-following error between a medium and long horizon. A 4-second horizon performs as well as an 8-second horizon, at a lower computational cost.

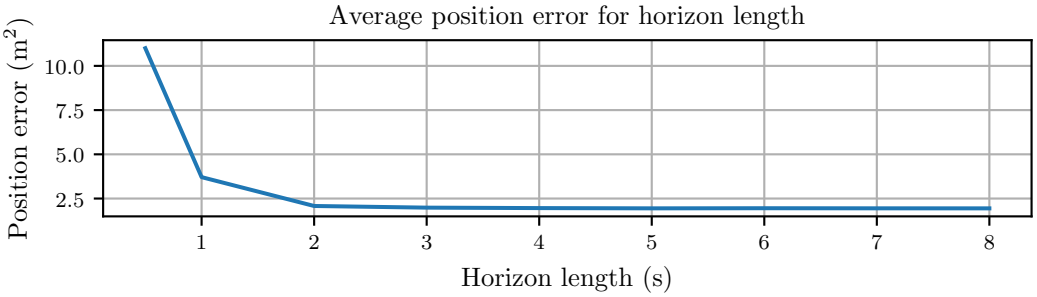


Figure 6.11: Target-following error vs horizon length.

6.2.4 Experiment 3: Replanning rate

This experiment investigates the effect of the replanning rate on the performance of the trajectory planner. Figure 6.12 shows the impact of the replanning rate on the success rate of the trajectory planner (the success rate was determined for 15 543 planning problems). In all cases, the success rate is near 100%, but the ability to generate the trajectories within real time degrades. Figure 6.13 provides additional insight. The planning time is not affected by the replanning rate. However, the allowed planning time changes, which affects the number of trajectories that can be planned within the time limit.

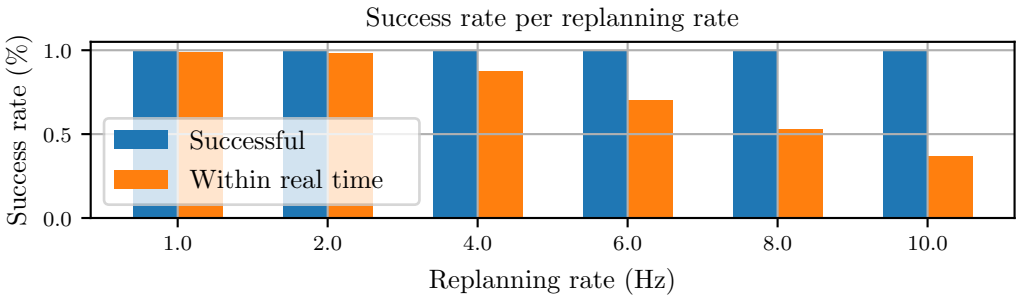


Figure 6.12: Success rate vs the replanning rate.

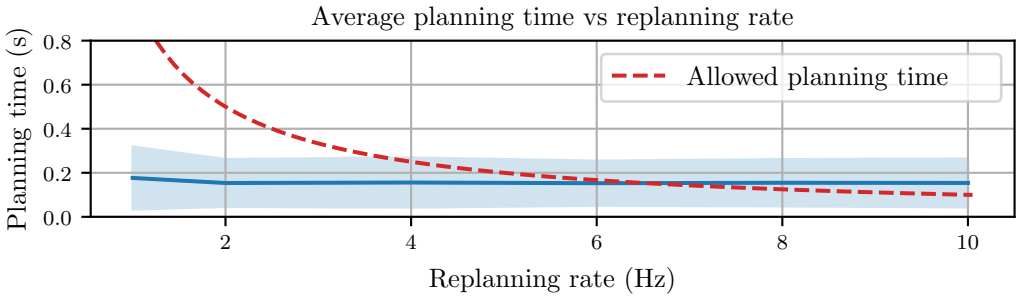


Figure 6.13: Planning time vs the replanning rate.

The target-tracking error is plotted against the replanning rate in Figure 6.14. The error decreases as the planning rate increases because the planner can react faster to a target changing its course. However, as explained above, the maximum replanning rate is limited by the planning time.

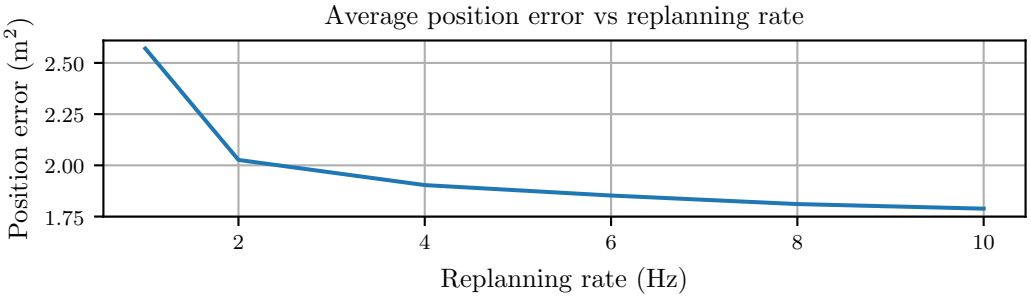


Figure 6.14: Objective function vs the replanning rate.

6.3 Summary

This chapter presented theoretical arguments and simulation experiments to determine the effect of specific design parameters, namely the planning resolution, the planning horizon, and the replanning rate, on the performance of the trajectory planner. The experiments show that it is necessary to balance all the parameters to achieve a practical system. The effect of each of the parameters is summarised as follows:

- **Planning resolution**: The higher the planning resolution, the finer level of control is possible, at the cost of longer computation time. The planning resolution should be high enough so that the maximum distance between the collocation points is smaller than the minimum safe separation distance between the UAV and obstacles.

- **Horizon length**: The longer the horizon length, the further into the future the trajectory planner can anticipate and plan for obstacles that will be encountered, but the higher the associated computational cost. The planning horizon should be at least long enough to allow the UAV to decelerate from its maximum velocity to a standstill (to avoid collisions with obstacles) or to accelerate from standstill to its maximum velocity (to match speed with and follow the target). The planning horizon should also be short enough so that computational effort is not wasted on planning too far into an uncertain future. Due to the inherent uncertainty in the target's behaviour, the target's predicted trajectory constantly changes, requiring the UAV trajectories to be constantly replanned. The upper bound for the planning horizon is therefore determined by the point where the diminishing improvement in the target-following performance does not warrant the increase in the computational cost.

- **Replanning rate**: The faster the replanning rate, the quicker the planner can react to changes in the target trajectory prediction. However, the replanning rate is limited by how fast the trajectory planner can plan the trajectories. The replanning rate should therefore be high enough to accommodate the rate at which the target trajectory changes, but low enough to accommodate the time it takes for the trajectory planner to execute.

The parameter selection case study also aided in selecting a suitable set of design parameters for the implementation. The selected parameters are summarised in Table 6.2. The maximum acceleration and velocity are based on the specifications of the AscTec Firefly UAV. The design parameters are based on the experiments and selected to achieve the best performance given the available computational power. These parameters are used in Chapter 7 to verify the performance of the trajectory planner. These parameters are specific to the current implementation and could be adjusted depending on the implementation and computational resources.

Table 6.2: Parameters for optimisation

Item	Value
Maximum acceleration	$4\ \mathrm{m\,s^{-2}}$
Maximum velocity	$15\ \mathrm{m\,s^{-1}}$
Planning resolution (t_s)	0.4 s
Horizon length	4 s
Replanning rate	4 Hz

Chapter 7

Simulation Results

In this chapter, the trajectory planner and target-following system are tested and evaluated using the simulation environment that was created for this purpose (as described in Section 5.4). First, the original system requirements are relisted, followed by the testing strategy to verify that the requirements are satisfied. A couple of examples are presented to illustrate and test the proposed trajectory planner and target-following system. After the examples are presented, the trajectory planner is tested independently in a variety of randomly generated scenarios (for the ground targets and physical environments described in Section 5.4). The results are analysed, and the strengths and weaknesses of the system are identified. Specific failure cases are also considered to gain insight into the reasons for planning failures.

7.1 System requirements

This chapter aims to verify that the original system requirements, as unpacked in Section 3.1, are satisfied. These system requirements are:

1. Plan collision-free trajectories for multiple UAVs.
2. The UAVs must follow a ground target.
3. The UAVs must avoid collisions with the environment and with one another.
4. The planner must be compatible with other components within a target-following system.
5. The trajectory planner should react to changes in the predicted target trajectory.
6. The planned trajectories should be dynamically feasible to ensure that the UAVs can execute them.

7.2 Requirements testing approach

The requirements testing approach can be divided into two parts. The first part presents a few examples to illustrate and test the trajectory planner and target-following system. The second part tests the proposed trajectory planner in a wide variety of randomly generated scenarios. This is done to obtain a quantitative analysis of how well the planner performs.

The first example tests the trajectory planner in an environment with no obstacles (the desert environment). This example provides a benchmark for the second example, where the same target is followed, now in an environment with obstacles (the forest environment). These examples show that the trajectory planner can plan trajectories to follow a target while avoiding collisions between UAVs and between UAVs and the static obstacles. The third example tests the complete target-following system in the Gazebo simulation using representative simulation models for the UAVs to verify that the trajectory planner can generate dynamically feasible collision-free trajectories and that the trajectory tracker can successfully execute the planned trajectories. This example also verifies that the planner is compatible with the other components within the designed target-following system.

Next, the trajectory planner is tested independently for a large number of randomly generated scenarios representing a range of physical environment types (desert, savanna, forest and urban) and a range of target behaviours (slow and fast, low agility and high agility). The performance of the trajectory planner is evaluated in terms of success rate, the target following error, the number of optimisation iterations, and time needed to compute the solution. These tests verify that the trajectory planner can operate in different environments, and to identify any shortcomings of the proposed trajectory planner.

7.3 Target following examples

As stated in the requirements testing approach, this section presents three examples to illustrate and test the trajectory planner and target-following system.

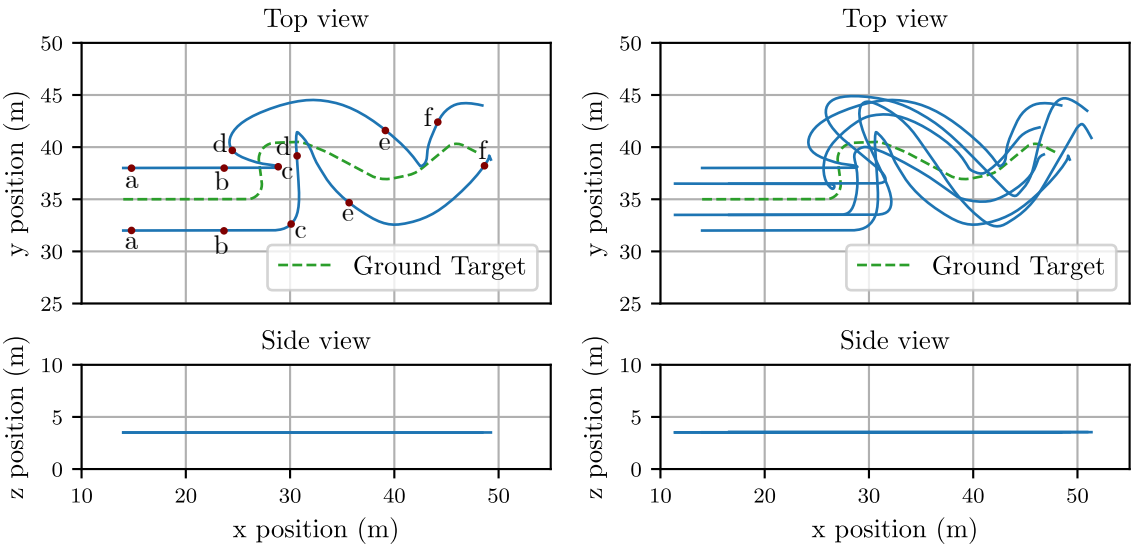
7.3.1 Example: No obstacles (desert environment)

This example presents a scenario in which the UAVs follow a moving target (high agility medium velocity target) through an environment with no obstacles (the desert environment). Figure 7.1 shows both the path of the target (green dashed line), as well as the planned paths of the UAVs (blue solid lines). Figure 7.1 (a) shows the scenario for two UAVs, and (b) shows the same scenario, but with six UAVs following the target. Some points are labelled on the graph, which will be discussed later in the section.

These trajectories were planned with the proposed trajectory planner described in Chapter 4. The trajectory of the target was not known in advanced. Therefore, the trajectory planner predicted the future target trajectory, planned the trajectories for the UAVs accordingly and repeated the process at the replanning rate. The figure shows that the planned trajectories stay next to the target, even without knowing the future intent of the target. The effects of not knowing the target trajectory in advance are visible in Figure 7.1 (a). Each time the ground target changes direction, the UAVs 'overshoot' the reference positions next to the target, but then quickly corrects for the various deviations.

Target following error

Figure 7.2 shows the position error relative to the reference position next to the target at every time-step, for the scenario illustrated in Figure 7.1 (a). Some key points are labelled, corresponding to the labels in Figure 7.1 (a). The two UAVs start at the



(a) 2 UAVs. (b) 6 UAVs.

Figure 7.1: Multiple UAVs following a high agility, medium speed target through the desert (obstacle free) environment.

correct positions relative to the target. Therefore, there are no errors at time-step zero. However, the target starts moving instantaneously, where the UAVs need to accelerate to match the velocity. The UAVs fall behind by time-step a, but recover and reduce the error by time-step b. At time-step c, the target rapidly changes direction, resulting in a large overshoot in terms of the target following error. However, the UAVs return closer to the ideal position by time-step d. The behaviour repeats for every target trajectory change in the rest of the trajectory.

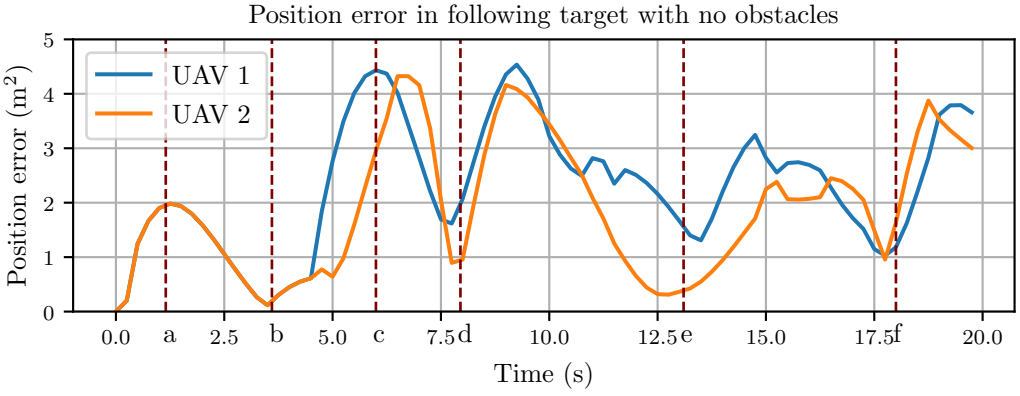
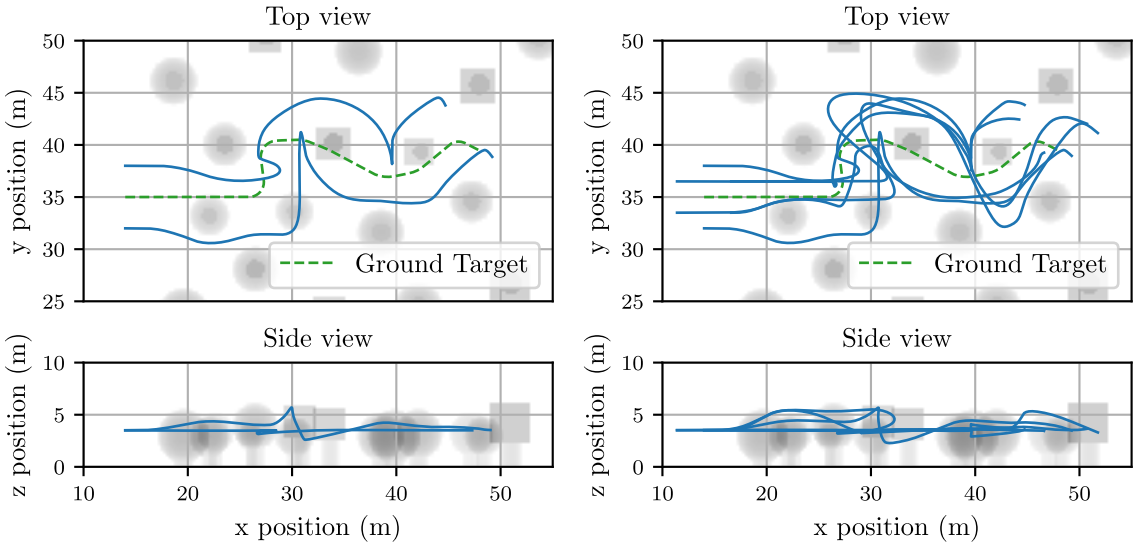


Figure 7.2: Position error in following the target for scenario with two UAVs (no obstacles).

7.3.2 Example: With obstacles (forest environment)

The previous example showed that the trajectory planner is capable of generating trajectories to follow a target with an unknown trajectory through an environment without any obstacles. This section expands on the previous example by following

the same target, now in an environment with static obstacles. Figure 7.3 shows the same target following scenario as in Figure 7.1, but this time in the forest environment. Similarly to the previous example, the UAVs still follow the ground target, but now the UAVs need to account for collisions with the environment. The side view shows that the trajectory planner also adjusts the vertical height of the UAVs to avoid collisions with each other and with the environment.



(a) 2 UAVs. (b) 6 UAVs.

Figure 7.3: Multiple UAVs following a high agility, medium speed target through the forest environment.

Collision avoidance

To ensure that the trajectories are indeed collision-free, Figure 7.4 shows the closest distance between any two UAVs, and the closest distance between any UAV and the static obstacles in the environment, at every time-step, for the scenario illustrated in Figure 7.3 (b). The red line indicates the enforced safety radius (the minimum allowed distance between UAVs). The graph shows that the minimum distance between UAVs stays above the enforced safety constraint.

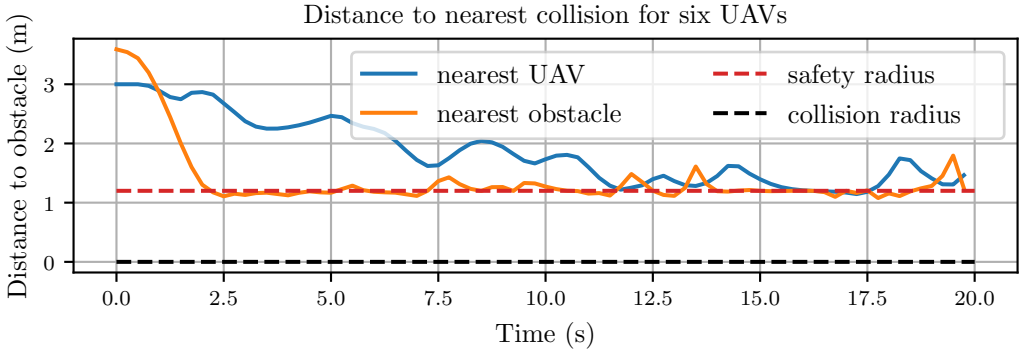
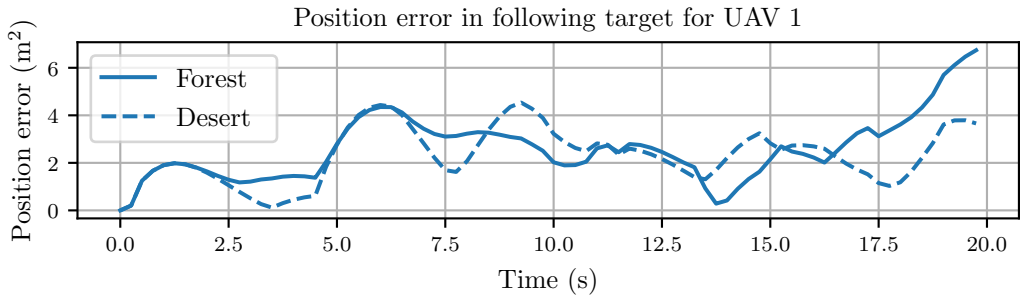


Figure 7.4: Distance to the nearest UAV and obstacle for the six UAVs following the target.

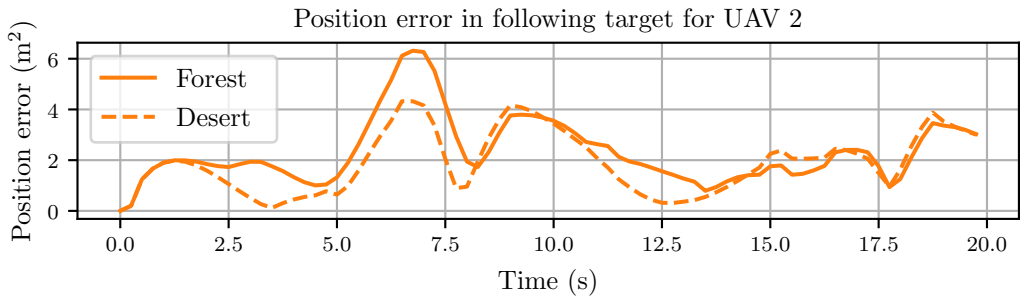
The minimum distance between the UAVs and the nearest obstacle stays mostly above the line, marginally dipping below the line at a few time instances. As described in Section 6.1.2, this is due to collisions only being checked at the collocation points. However, the planning resolution is high enough that the violation is minimal. The violation is also significantly smaller than the safety radius.

Target following error

Figure 7.5 shows the target following error for both the desert environment and the forest environment shown in Figure 7.1 (a) and Figure 7.3 (a). In both cases the same target is being followed. The graphs show that, in general, the target follow ability degrades when there are obstacles, as the UAVs need to avoid the obstacles.



(a) UAV 1



(b) UAV 2

Figure 7.5: Position error for the two UAVs following the ground target. The graph shows both the error for the desert and the forest environment.

7.3.3 Example: Target following system

To verify that the target-following system is viable for practical implementation, the complete target-following system, described in Chapter 5, was implemented in ROS and tested using the Gazebo simulation environment. The simulation environment includes the trajectory planner, the trajectory execution components, the target trajectory prediction, the environment model, the inter-component communication, and the representative models of the UAVs and their flight controllers. A test scenario is created in which four UAVs are tasked to follow a randomly moving target.

Results

Figure 7.6 shows the planned and the executed trajectory for each of the agents. As before, the proposed trajectory planner predicts the target trajectory, plans the corresponding UAV trajectories and repeats the process at the specified replanning rate. However, in this example, the trajectory planner publishes the planned trajectories to the trajectory trackers over a ROS topic at the replanning rate (as described in Chapter 5). The different trajectory trackers control the models of the UAVs within the Gazebo simulation. The graph shows that the executed paths closely align with the planned paths. Some key points are labelled, which will be discussed later in the section.

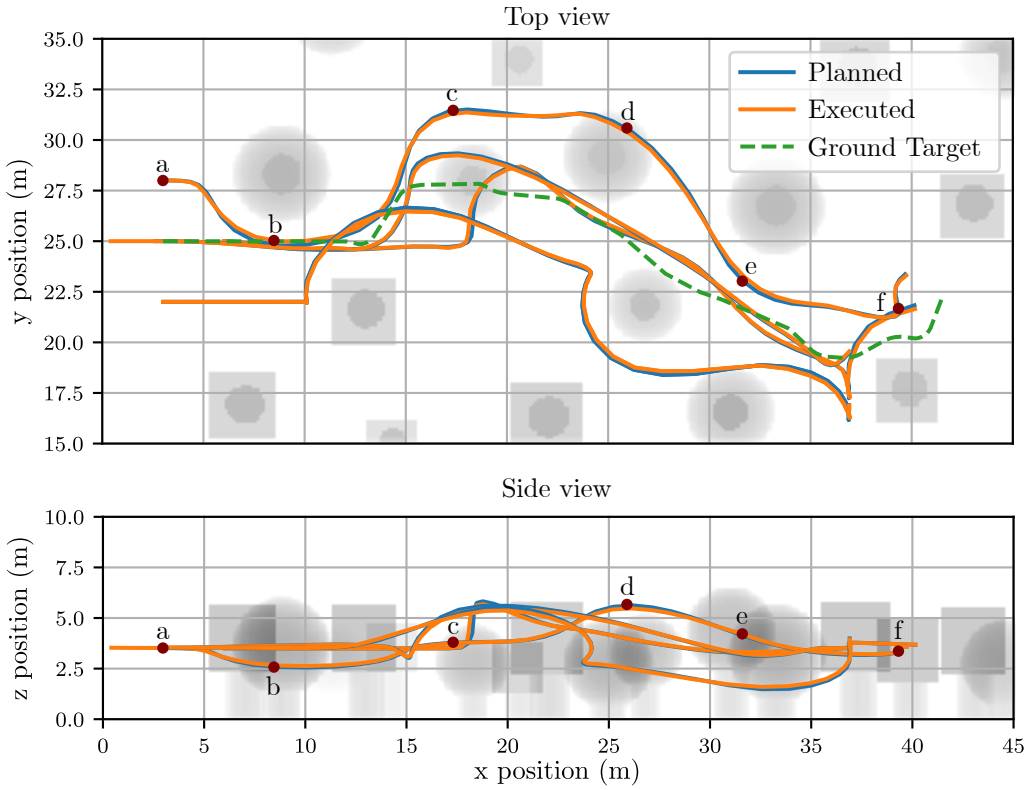
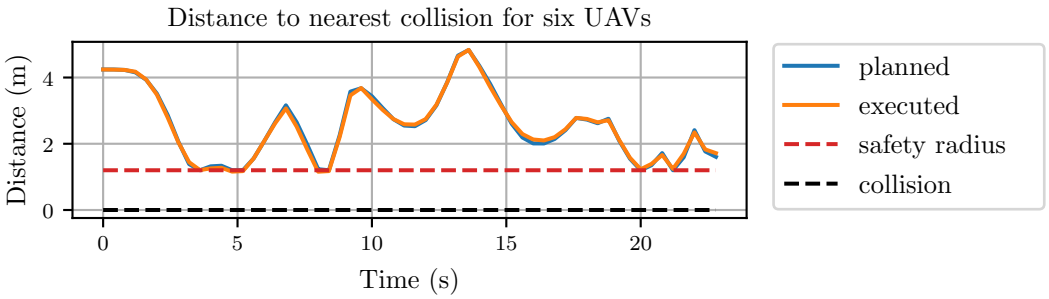


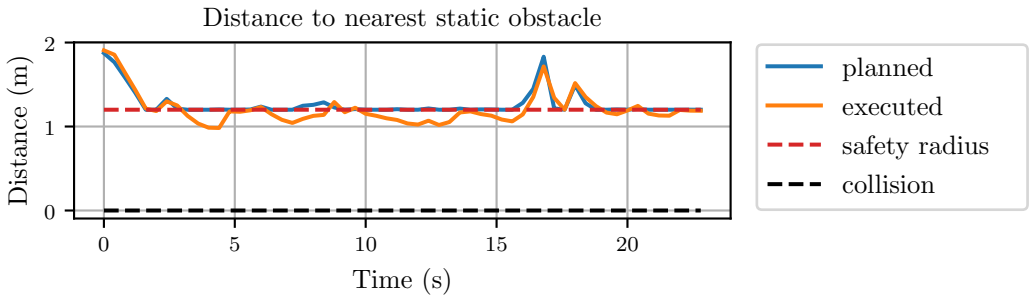
Figure 7.6: Planned and executed trajectories for four UAVs following a moving ground target.

Collision avoidance

It is possible that the deviations between the planned and the executed trajectories can violate the collision constraints. Therefore, Figure 7.7 (a) shows the closest distance between any two UAVs, and Figure 7.7 (b) shows the closest distance between any UAV and the static obstacles, for both the planned and executed trajectories. As before, the planned trajectories satisfy the collision constraints, both in terms of inter-UAV collisions, and collisions between UAVs and static obstacles. The executed trajectories sometimes violate the safety radius, especially between UAVs and obstacles, but sufficient margin is still provided between the safety distance and the actual collision distance. The rest of the section will further analyse and quantify the trajectory tracking error.



(a) Minimum distance in between the UAVs



(b) Minimum distance between UAVs and static obstacles

Figure 7.7: Collision clearance for the target following system, both in terms of minimum distance between UAVs and obstacles, and in between UAVs.

Trajectory tracking error

The performance of the trajectory tracker is measured by how much a UAV deviates from its planned trajectory. This section will evaluate the trajectory tracking performance based on the cross-track error, along-track error, and vertical error, as illustrated in Figure 7.8.

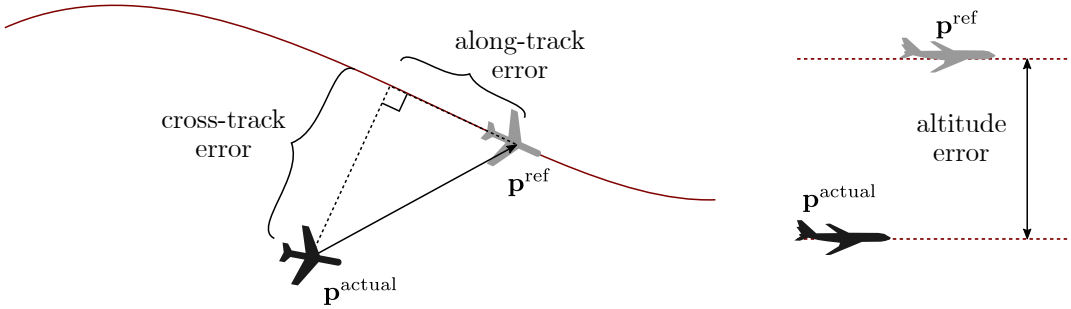


Figure 7.8: Definition of cross-track, along-track and vertical error for measuring the performance of the trajectory tracker.

The errors are defined as:

- **Along-track error**: The along-track error is how far ahead or behind the UAV is of its ideal reference position, projected along the intended path, at each time-step.
- **Cross-track error**: The cross-track error is the error between the actual and reference position, projected onto a vector perpendicular to the reference path.

- **Altitude error**: The altitude is the UAV's vertical position error.

Tracking analysis: UAV 2

This section discusses the trajectory tracking error in more detail. Figure 7.9 shows the tracking error for UAV 2 against time. To avoid 'cluttering' the graph, the position errors of the other UAVs are not included, but similar behaviours were observed. Some key points are labelled, corresponding to the positions indicated in Figure 7.6. At position (a) there is a peak in the along-track error. This shows that the UAV momentarily falls behind as it starts executing the trajectory, accelerating from standstill to match the target velocity. Figure 7.6 further shows that the other peaks (b)-(f) correspond to positions where the UAV is busy turning (or changing direction).

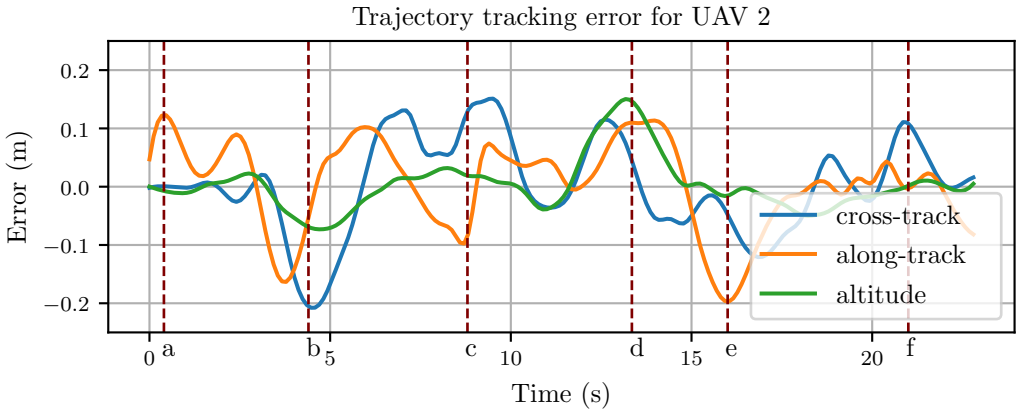


Figure 7.9: Trajectory-tracking error for UAV 2 following the ground vehicle.

Figure 7.10 summarises the trajectory tracking error for each of the UAVs. The original position error in the x_W and y_W plane is rotated to an along-track error and a cross-track error. The graph also shows the vertical error versus the cross-track error. All the errors are centred around the origin, which shows that there is no bias error for the tracking. The furthest trajectory deviation is approximately 20 cm, which corresponds to the violation observed in Figure 7.7. This shows that although the trajectory tracker can execute the planned trajectories within a margin of error (which is significantly less than the selected safety radius), the target following system could potentially benefit from tighter integration between the trajectory planner and the trajectory trackers.

7.4 Trajectory planning testing

The previous section presented examples which showed that the trajectory planner is capable of generating trajectories for the UAVs to fly next to a ground target. This section expands on the previous examples by testing the trajectory planner in a wide variety of randomly generated target-following scenarios. These simulations quantify how well the trajectory planner performs in different environments, with different numbers of UAVs and different ground targets.

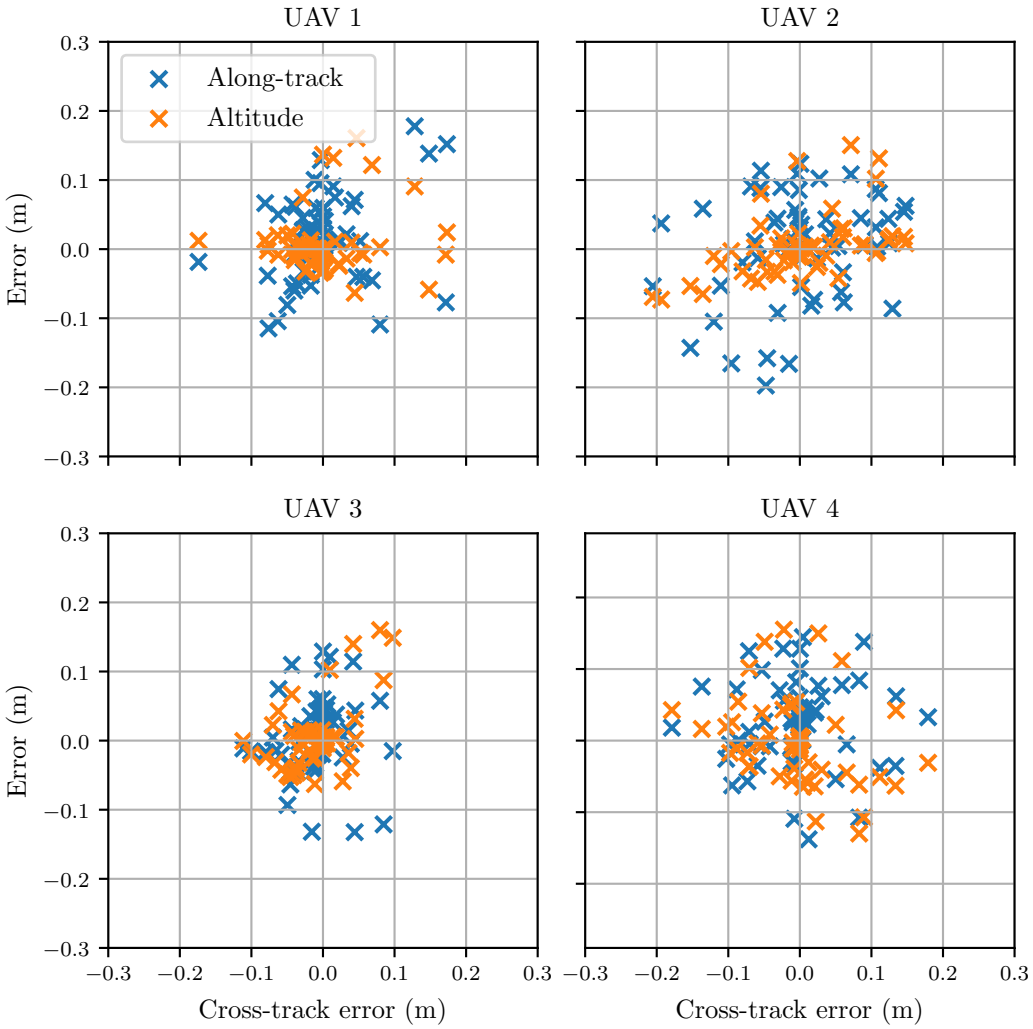


Figure 7.10: Tracking error of each UAV while executing the trajectory. The graph shows both the along-track error and altitude error against the cross-track error.

7.4.1 Simulation test setup

The simulation environment described in Section 5.4 was used to generate random target-following scenarios. The four physical environments were used, with 60 randomly generated ground targets ranging from fast to slow, and high agility to low agility. Every target was used in every environment, resulting in $4 \times 60 = 240$ combinations. Each of the combinations was tested for the number of UAVs ranging from 1 to 8, resulting in $240 \times 8 = 1920$ target following scenarios. Due to the ongoing replanning strategy, the simulation tests resulted in 164 736 optimal control problems for the trajectory planner to solve.

7.4.2 Simulation test results

This section presents the results of the simulations described in the test setup. The results are discussed in terms of success rate, target-following error, planning time and optimisation iterations. The success rate is how many of the individual trajectory planning problems were successfully solved. The target-following error measures how

much the trajectories deviate from the reference position relative to the target. The planning time is the total time required to plan a trajectory at each planning iteration. The optimisation iterations indicate the computational effort, independent of specific hardware and software implementation.

7.4.3 Success rate

The breakdown of the success rate of the trajectory planner is shown in Figure 7.11. The figure shows the success rate for the different number of UAVs, categorised into the environments type and the target speed and agility.

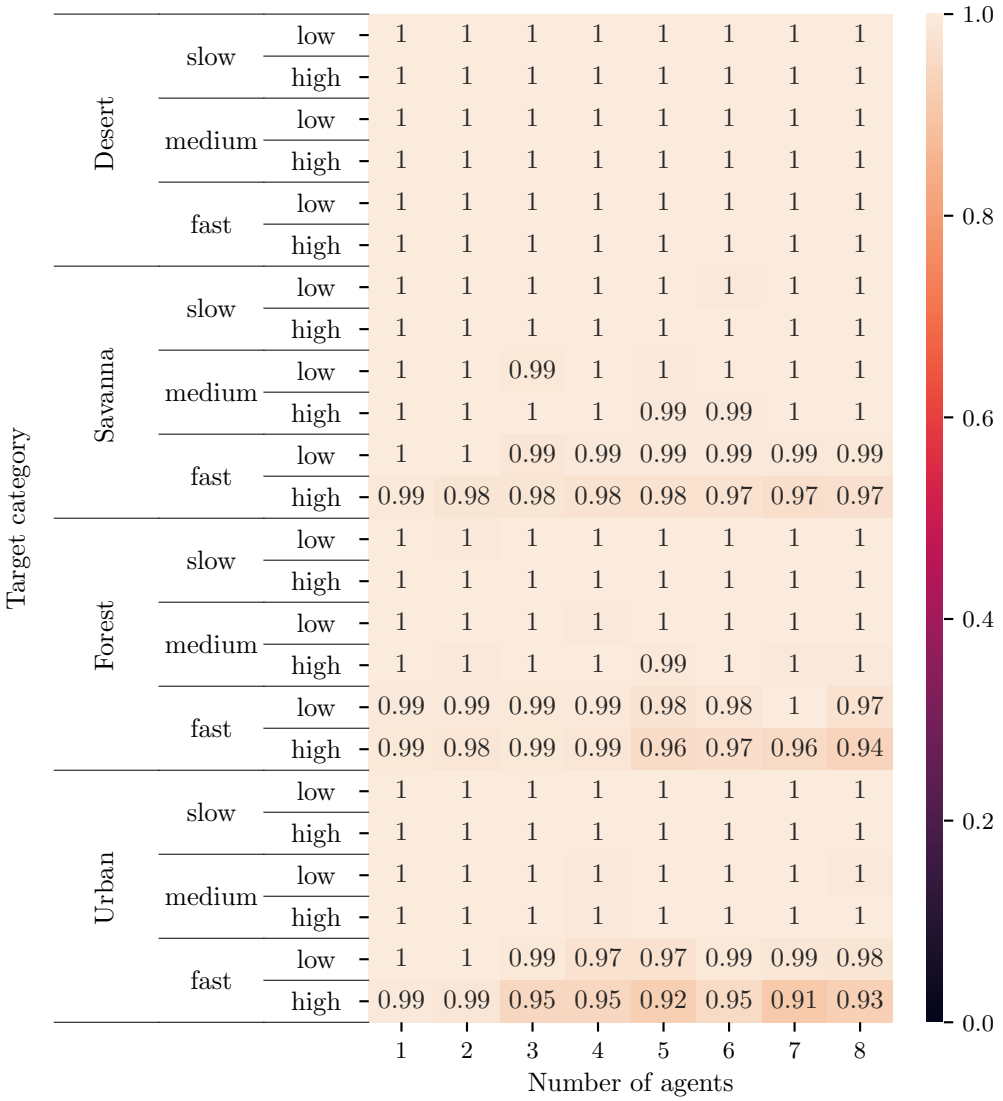


Figure 7.11: Breakdown of success rate per number of agents in different environments and target types (slow, medium and fast moving as well as low and high agility).

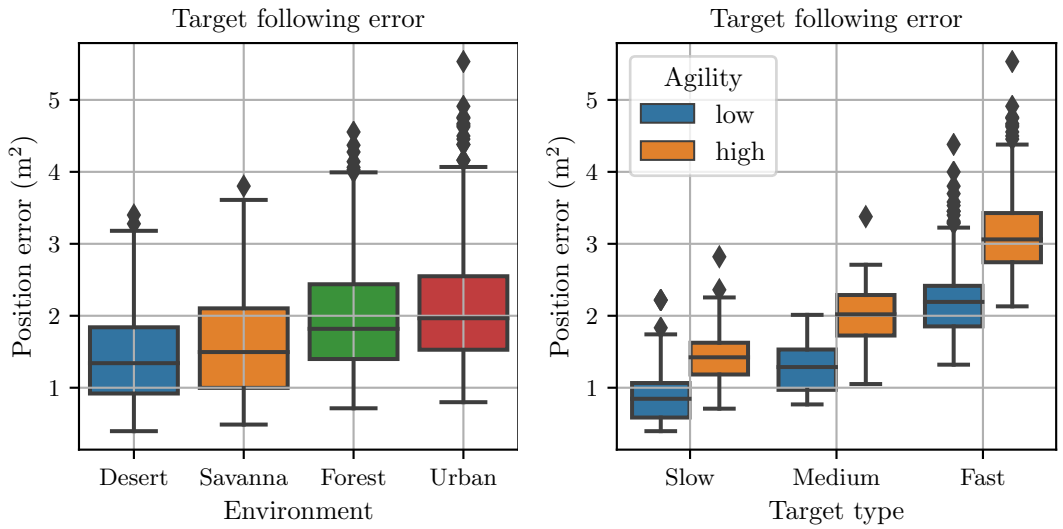
The trajectory planner achieved a near 100% success rate across all the physical environments, given a slow or medium velocity target. The success rate only notably degrades (down to a 91% success rate) for scenarios with a fast-moving, high agility

ground targets in very cluttered environments when planning for a high number of UAVs. Specific failure cases will be further discussed in 7.5.1.

It is important to remember that a failed planning iteration does not imply a failure of the overall target-following system. It only means that a single iteration of the replanning strategy is skipped, thereby degrading the target-following performance. In none of the simulated scenarios did a failed planning iteration result in a collision between UAVs or a collision with static obstacles.

Target-following error

Figure 7.12 summarises the average target-following error for the 1920 target-following scenarios. Figure 7.12 (a) shows the target-following error for each of the environments. The target following ability of the trajectory planner degrades in more cluttered environments, such as the forest and urban environment. This is because the trajectory planner adjusts the paths to avoid collisions. The target-following error for different target types is presented in Figure 7.12 (b). The target-following error is the lowest when following a low agility, slow target. The performance degrades as the agility and the velocity of the target increases. The graph shows that both the agility and the velocity of the target have an impact on the target-following ability of the trajectory planner. These graphs verify that the trajectory planner can plan trajectories to follow the ground target in a wide variety of scenarios.



(a) Error vs environment

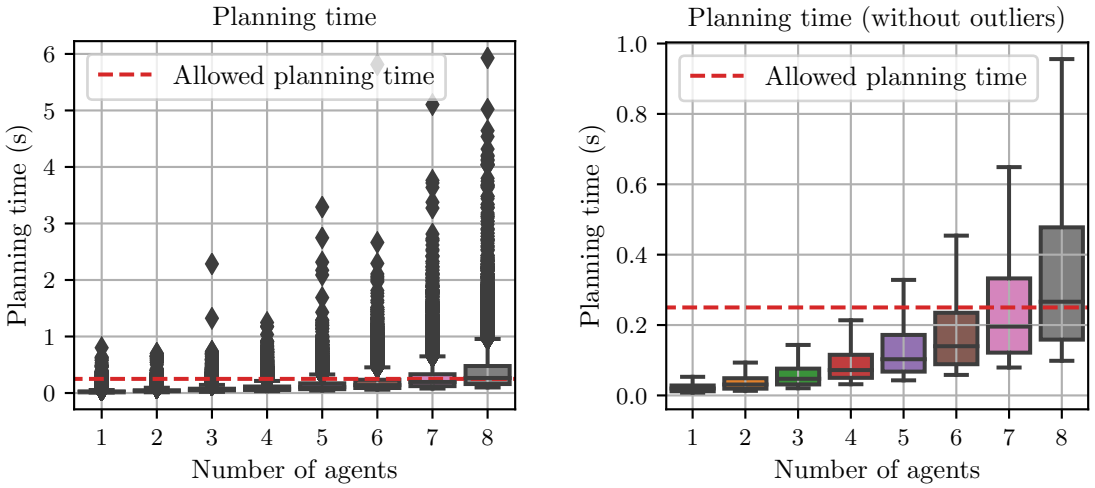
(b) Error vs target type

Figure 7.12: Position error for the randomly generated target-following scenarios.

Planning time

To evaluate the ability of the proposed trajectory planner to plan trajectories in real time, the distribution of the trajectory planner's execution time per iteration is analysed and compared to the planning interval. Figure 7.13 shows the planning time as a function of the number of agents for the current implementation. Figure 7.13 (a) includes the outliers in terms of the planning time. The outliers are either the result of

the algorithm needing more iterations to converge, or disturbances from the operating system, which will be further discussed below.[1] Figure 7.13 (b) is included with the outliers removed, as it shows the general trend in planning time more clearly. The red line indicates the maximum allowed planning time due to the replanning interval of the replanning strategy. The graph shows that, for up to four UAVs, the planning time is within the maximum allowed time. The graph also shows that planning for more agents within the allocated time is within reach. The 75th percentile of the planning time for six UAVs is within the allowed planning time. However, the planning time does follow an exponential trend as the number of agents increases, which corresponds with the theory presented in Section 2.3. The computational complexity significantly increases with the number of agents, as the size of the search space increases. Real-time planning for a larger number of UAVs could potentially be achieved by using dedicated hardware with more computational power and by optimising the software implementation, or by investigating a distributed approach.



(a) With outliers.

(b) Without outliers.

Figure 7.13: Total planning time for different numbers of agents.

Optimisation iterations

For the trajectory planner to be viable for real-time implementation, it must be able to plan trajectories reliably within a maximum allowed planning time. However, using planning time as a performance metric introduces some challenges. First, planning time is dependent on the specific hardware and software implementation. This means that the timing performance can be improved by increasing the computational power of the hardware or by optimising the software implementation. Secondly, the timing performance of the trajectory planner when executed on a desktop computer is not necessarily an accurate indication of the timing performance that could be achieved on dedicated hardware. If a planning iteration takes too long, it could be due to the posed trajectory planning problem being difficult to solve, or because computational time was 'stolen' by other background processes running on the same computer.

[1] Outliers are identified with the interquartile range rule [82].

It is important to determine whether the trajectory planner will sometimes take longer to execute independently of its hardware or software implementation. By looking at the number of optimisation iterations, it is possible to identify scenarios in which the planner takes longer to execute. Figure 7.14 plots the number of optimisation iterations against the absolute planning time (the colour indicates the number of UAVs) for the successful iterations. Firstly, the graph shows that there is a correlation between the two variables, which shows that the optimisation iterations can be indicative of the planning time.

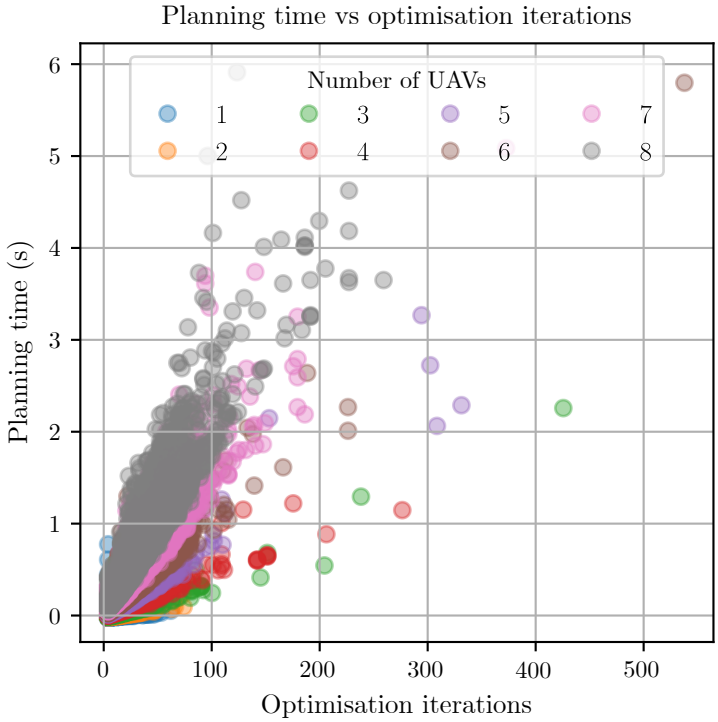


Figure 7.14: Correlation between planning time and the number of iterations.

Secondly, the graph shows most of the points are clustered in the lower-left corner, which indicates that most of the time the trajectory planner exhibits both a short planning time and a low number of optimisation iterations. However, there are several outliers that exhibit both longer planning times and a larger number of optimisation iterations. Therefore, most of the time, the scenarios result in well-posed optimisation problems for which the optimisation algorithm converges quickly and with a low number of iterations. However, sometimes the scenario results in an ill-posed optimisation problem for which the optimisation algorithm takes longer to converge and needs a larger number of iterations. Some specific cases in which the optimisation algorithm takes longer to converge will be discussed in Section 7.5.1.

7.5 Results analysis

The presented results highlight a few important aspects regarding the trajectory planner. Firstly, the trajectory planner performs best when the target is travelling at a low or medium velocity, or if there are relatively few agents. The type of environment also impacts performance. The performance degraded as the number of obstacles increased. From a practical perspective, this makes sense. A scenario in which two UAVs should

avoid a few trees is much easier to solve than generating paths for eight UAVs flying through a narrow gap. As the number of agents and obstacles increases, the amount of free space for each agent to move decreases, demanding more complex trajectories from the trajectory planner.

If there is a situation in which a single UAV may fail, having multiple UAVs will only amplify any problems that may occur. The more UAVs that are operating in the environment, the more likely it is that a single UAV may find itself in a situation in which the trajectory planner fails. This corresponds with Figure 7.11 showing that the success rate decreases as the number of UAVs increases.

7.5.1 Failure cases

Through manual inspection of the failure cases, two scenarios were identified in which the trajectory planner struggled to converge to a solution. In some cases, this resulted in the planner taking more iterations than expected. In more extreme cases, this resulted in a failed planning iteration. The two scenarios are discussed below, followed by possible solutions to address the problems.

Scenario 1

The grid-search strategy does not consider the dynamics of the UAVs when planning an initial estimate. It could provide an initial estimate to the trajectory optimisation algorithm from which it cannot converge to a collision-free, dynamically feasible solution. To illustrate a scenario in which the strategy can be undesirable, Figure 7.15, presents two possible initial estimates for a planning scenario. The one path "snakes" between obstacles, whereas the other path takes a more conservative approach around obstacles. In some cases where the trajectory planner failed, it would have been possible for the trajectory planner to plan trajectories that fly over the obstacles. Yet, the initial estimate creates a path that 'snakes' through the obstacles. The grid search strategy only considers which path is shorter and selects it. This is the result of using a different cost function for the initial trajectory generation than for the optimal control phase. However, it may not be dynamically feasible for the UAV to travel between the obstacles, at least not at the velocity required to follow the target. At low velocities, this does not pose a problem, but it fails when the UAVs are travelling at high velocities.

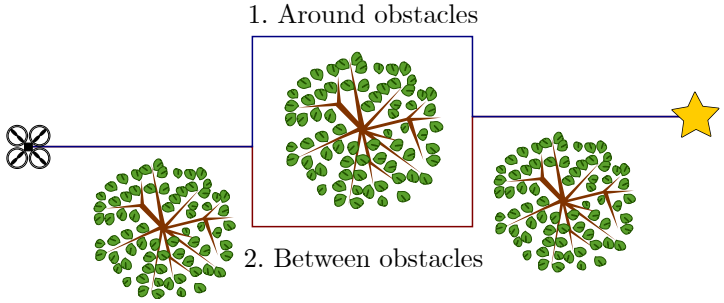


Figure 7.15: The grid search finding a winding path between obstacles when a longer, but simpler path exists.

Scenario 2

The second scenario is also a result of the initial estimate neglecting the dynamics of the UAVs. The failure occurs when the target rapidly changes direction in front of an obstacle, which is often the case if the target is following a high-agility target. Figure 7.16 illustrates such a scenario. At time-step t_k, a UAV is following a target heading towards an obstacle and plans a trajectory around the obstacle. However, at time-step t_{k+1}, the UAV turns away in front of the obstacle. At that stage, the UAV is already travelling at a high velocity next to the target. The grid-search strategy calculates an initial estimate that requires the UAV to stop and travel in the opposite direction. If the velocity of the target is too high, the optimisation strategy will fail as it cannot generate a trajectory from the initial estimate that satisfies all the constraints.

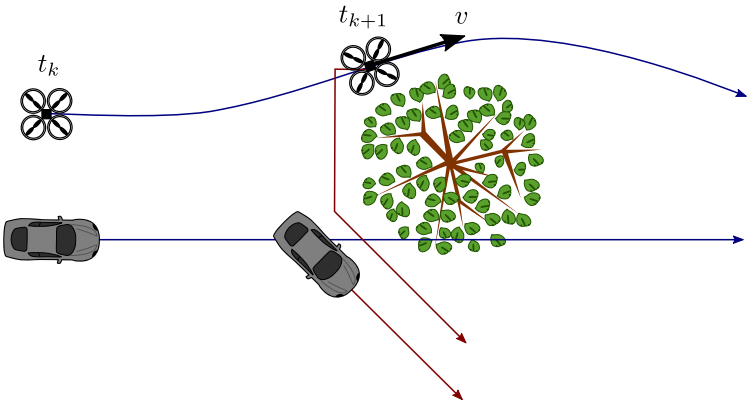


Figure 7.16: Scenario in which the ground target quickly changes direction right before an obstacle.

As before, when the UAV is travelling at a low velocity, the initial estimate does not create a problem (it may result in the optimisation algorithm taking a few extra iterations to converge). However, the initial estimate creates a problem when the UAVs are travelling at high velocities.

Possible solutions

Both failure cases are a result of the initial grid-search strategy that neglects the dynamics of the UAVs. A possible solution is to incorporate the dynamic constraints into the search strategy. However, this would increase the computational complexity of generating the initial estimates as it would result in a higher-dimensional search problem, which could be solved using a sampling-based planner such as the RRT*.

Another possible solution could be to use the solution of the previous planning iteration as an initial estimate for the next iteration. This was tested to some extent, but discarded as it generally resulted in less optimal trajectories, especially when following a highly agile target. It is, however, less computationally expensive, if the target does not change direction.

7.6 Summary

This chapter presented simulation results to verify that the trajectory planner satisfies the original system requirements. First, experiments were presented to illustrate and

test the trajectory-planner and target-following system. Secondly, the trajectory planner was tested in a wide variety of randomly generated target-following scenarios. The results were analysed in terms of success rate, target-following error, planning time, and optimisation iterations. The trajectory planner is able to plan trajectories for multiple rotary-wing UAVs to cooperatively follow a moving ground target while avoiding collisions with the environment and between UAVs, as well as obeying the dynamic constraints of the UAVs. The trajectory trackers on the UAVs are able to follow the planned trajectories, with a slight violation of the planned minimum separation distance, but with a sufficient margin of safety. Therefore, the proposed planner satisfy the original system requirements.

Chapter 8

Conclusion

The project aimed to develop a trajectory planner for multiple rotary-wing UAVs to cooperatively follow a moving ground target while avoiding collisions with the environment and between UAVs, as well as obeying the dynamic constraints of the UAVs. This chapter concludes the report by providing a *summary of work*, some *key findings* as well as highlighting *future research opportunities*.

8.1 Summary of work

A literature review was performed on trajectory planning and target-following approaches. The study revealed that there is limited published research on cooperative target following with multiple rotary-wing UAVs while avoiding obstacles. The review also showed that planning for multiple agents while including their dynamics is a large and non-trivial planning problem. Due to the nature of target following, the trajectories must regularly be replanned in real time, as the predicted target trajectory may change.

Based on the literature review, optimal control was selected as the most suitable approach for the target-following problem. Optimal control was investigated, and trajectory optimisation (specifically a direct collocation technique) was selected to transcribe the problem. The design choices were motivated with background research on trajectory optimisation and non-linear optimisation. Euclidean Signed Distance Fields (ESDFs) were selected to represent the environment. A replanning strategy similar to Model Predictive Control (MPC) was selected to plan trajectories for a target with unknown intent.

The architecture and design of the proposed trajectory planner was presented. The constraint and objective functions for the target-following problem were formulated. A grid-search strategy to provide an initial estimate of the solution was also designed. The chapter presented the replanning strategy and introduced the concept of recursive feasibility. The trajectory planner was designed to ensure that the UAVs hover at the end of the planning horizon, which ensures safety in the event of a failed planning iteration. An example was used to illustrate how the trajectory planning problem is iteratively solved with trajectory optimisation.

The trajectory planner was combined with other components through a ROS architecture to create a target-following system. A simulation environment was constructed, which was used to test the trajectory planner in a wide variety of scenarios. The environment includes the maps of the environment, the targets that are being followed, and the representative UAV simulation model.

The impact of specific design parameters, namely the planning resolution, the planning horizon, and the replanning rate, was investigated from both a theoretical standpoint and through simulation experiments. The experiments showed that it is necessary to balance all the parameters to achieve a practical system.

Finally, the simulation results were presented to verify that the trajectory planner satisfies the original system requirements. First, experiments were presented to illustrate and test the trajectory-planner and target-following system. Secondly, the trajectory planner was tested in a wide variety of randomly generated target-following scenarios. The results were analysed in terms of success rate, target-following error, planning time, and optimisation iterations.

8.2 Key findings

The proposed trajectory planner can plan trajectories for multiple rotary-wing UAVs to cooperatively follow a moving ground target while avoiding collisions with the environment and between UAVs, as well as obeying the dynamic constraints of the UAVs. The trajectory trackers on the UAVs can follow the planned trajectories, with a slight violation of the planned minimum separation distance, but with a sufficient margin of safety. Therefore, the proposed planner satisfy the original project aim.

The project showed that the target-following problem is an interesting hybrid between a trajectory-planning problem and a control problem. It can be viewed as a trajectory-planning problem, as planning is needed to avoid collisions. However, it is not a classic trajectory-planning problem with a goal position at the end of the trajectory in mind. Instead, there is a desired position at every time-step. In that sense, the target-following problem is a control problem with a reference command at each time-step. However, in the more traditional autonomous robotics case, the control strategy is not developed to avoid collisions but instead handled by a higher-level planner.

Optimal control proved to be a powerful way to formulate the target-following problem, as it makes provision for an objective function as well as constraints. Modern NLP solvers proved powerful in solving large planning problems. However, the results showed that providing the solver with a good initial estimate is essential. The combination of a search algorithm with the optimisation strategy proved to be powerful. However, the shortcomings of neglecting the dynamics from the initial estimate become apparent when the UAVs are travelling at higher velocities. Future projects could investigate including some dynamic constraints when generating the initial estimate.

The receding horizon replanning strategy proved to be a powerful strategy to react to changes in the predicted target trajectory and to replan the UAV trajectories. Enforcing a constraint that the UAVs should come to a standstill at the end of the prediction horizon proved essential to ensure the planning strategy is safe. It ensured that the trajectory planner could not plan trajectories that would lead to future trajectory planning problems that could not be solved. It also provided a safety measure should the optimisation strategy fail to generate a solution.

8.3 Future research opportunities

This section presents suggestions for future work related to the project. The recommendations are divided into two categories. The first category focuses directly on the *trajectory-planning* problem. The second category focuses on projects in the bigger problem, needed to create a complete *target-following system*.

8.3.1 Trajectory planning

Initial estimate of solution

As discussed in Section 7.5.1, the trajectory planner could potentially benefit from considering the dynamics of the UAVs when generating the initial estimate of the solution. Therefore, future projects could consider alternative strategies for providing an initial estimate of the solution for the optimisation algorithm.

Different optimisation strategies

The project currently uses IPOPT, which is a general NLP library. These types of libraries focus on general NLP problems in different domains. The algorithm is not specifically designed for motion planning problems. Stella *et al.* [83] have shown that significant performance and convergence gains can be achieved if the algorithm is specifically designed with motion planning in mind. They proposed the PANOC (Proximal Averaged Newton-type method for Optimal Control) algorithm, which is a fast solver for non-linear optimal control problems. Such algorithms could be considered for future implementation.

In some cases, it is possible to decouple a complex optimisation problem into simpler problems which are solved sequentially. A popular strategy is to make use of the Alternating Direction Method of Multipliers (ADMM). Van Parys and Pipeleers [13] have shown that this strategy could significantly improve the performance of multi-agent trajectory optimisation problems. Such a strategy could be considered to improve the scalability for a larger number of agents.

Avoid dynamic obstacles

The current implementation only considers static obstacles in the environment and not dynamic obstacles. Future research could expand the problem to include dynamic obstacles. As it is quite computationally expensive to generate the ESDFs, a potential strategy could be to handle dynamic obstacles separately from static obstacles. Dynamic obstacles could potentially be treated as an additional set of constraints on the optimisation problem.

Incorporate uncertainty

The current system does not incorporate any uncertainty into the trajectory-planning problem. In reality, uncertainty is present throughout the system. There is uncertainty regarding the states of the agents, the state of the target, and the environment. The robustness of the algorithm could be improved by incorporating the uncertainties into the trajectory-planning problem. For instance, if there is much uncertainty regarding the position of the ground target, the weighting of the target-following objective could

be lower, as it does not make sense to allocate resources to follow a target with zero error if you do not know exactly where it is.

Consider a complete method

Trajectory optimisation achieves its fast performance by calculating the local solution to the optimal control problem. This means that there may be a loss of optimality in the solution. However, determining the true cost of using the local solution is difficult to measure, as the true solution is not known. By implementing a complete method such as mixed-integer programming, or by having random restarts, could provide a more systematic approach to verifying the performance, as the true optimal solution is known.

8.3.2 Target following system

Map generation

For the trajectory planner to work in an unknown environment, an online map generation system is needed. It would be necessary to generate a map of the environment from fusing information of multiple sensors distributed over multiple agents. Future research could aim to generate a map of the environment from multiple sensors. It could involve investigating alternative ways to represent an environment within an optimal control problem.

Target state estimation

The trajectory planner relies on knowing the current state of the ground target as well as predicting its future states. For a complete target-following system, it is necessary to estimate the states from sensors. Estimation could be achieved by placing an odometry sensor directly on the ground target, or by using input from cameras on the UAVs. This estimation model could also be used to provide a better prediction of the target trajectory for the replanning strategy. Future projects could investigate methods to estimate and predict the states of the ground target.

8.4 Reflection

In many ways, the experience of implementing the trajectory planner in this project can be summarised by this extract from *Underactuated Robotics* by Tedrake [68] on trajectory optimisation:

> As you begin to play with these algorithms on your own problems, you might feel like you're on an emotional roller-coaster. You will have moments of incredible happiness – the solver may find very impressive solutions to highly non-trivial problems. But you will also have moments of frustration, where the solver returns an awful solution, or simply refuses to return a solution (saying "infeasible"). The frustrating thing is, you cannot distinguish between a problem that is actually infeasible, vs. the case where the solver was simply stuck in a local minima.

> So the next phase of your journey is to start trying to "help" the solver along. There are two common approaches.
>
> The first is tuning your cost function – some people spend a lot of time adding new elements to the objective or adjusting the relative weight of the different components of the objective. This is a slippery slope, and I tend to try to avoid it (possibly to a fault; other groups tend to put out more compelling videos!).
>
> The second approach is to give a better initial guess to your solver to put it in the vicinity of the "right" local minimal. I find this approach more satisfying, because for most problems I think there really is a "correct" formulation for the objective and constraints, and we should just aim to find the optimal solution.

In this project, trajectory optimisation proved to be a powerful technique for quickly solving large optimal control problems. The technique is capable of quickly generating complex collision-free trajectories for multiple UAVs in a 3D domain. This is a problem that many established techniques cannot solve. However, the performance is very dependent on the problem formulation and the initial estimate of the solution. There are seemingly similar ways to formulate the objective and constraint functions, yet the one formulation converges, and the other does not.

In many ways, this is analogous to the current state of robotics described in Section 1.4. In some cases, robotics have exceeded our expectations, yet fall short in other. However, through continuous refinement and research, both trajectory optimisation and robotics are becoming increasingly powerful.

Bibliography

[1] A. Birk, "What is Robotics? An Interdisciplinary Field Is Getting Even More Diverse [Education]," *IEEE Robotics Automation Magazine*, vol. 18, no. 4, pp. 94–95, Dec. 2011. DOI: 10.1109/MRA.2011.943235.

[2] M. Rahul, "Review on Motion Capture Technology," *Global Journal of Computer Science and Technology: F Graphics & vision*, vol. 18, no. 1, pp. 1–5, 2018.

[3] X. Zhou, S. Liu, G. Pavlakos, V. Kumar, and K. Daniilidis, "Human Motion Capture Using a Drone," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2027–2033, 2018. DOI: 10.1109/ICRA.2018.8462830.

[4] L. Xu, Y. Liu, W. Cheng, K. Guo, G. Zhou, Q. Dai, and L. Fang, "FlyCap: Markerless Motion Capture Using Multiple Autonomous Flying Cameras," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 8, pp. 2284–2297, 2018. DOI: 10.1109/TVCG.2017.2728660.

[5] T. Nägeli, S. Oberholzer, S. Plüss, J. Alonso-Mora, and O. Hilliges, "Flycon: Real-time environment-independent multi-view human pose estimation with aerial vehicles," in *ACM Transactions on Graphics*, vol. 37, 2018, pp. 1–14. DOI: 10.1145/3272127.3275022.

[6] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed Distance Fields: A Natural Representation for Both Mapping and Planning," *Robotics: Science and Systems*, 2016.

[7] R. Fonseca and W. Creixell, "Tracking and following a moving object with a quadcopter," *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2017*, 2017. DOI: 10.1109/AVSS.2017.8078463.

[8] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006, pp. 1–826, ISBN: 0521862051. DOI: 10.1017/CBO9780511546877. arXiv: arXiv:1011.1669v3. [Online]. Available: http://planning.cs.uiuc.edu/.

[9] P. D. Nguyen, C. T. Recchiuto, and A. Sgorbissa, "Real-time path generation for multicopters in environments with obstacles," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-Novem, pp. 1582–1588, 2016. DOI: 10.1109/IROS.2016.7759256.

[10] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS—A modular gazebo MAV simulator framework," in *Studies in Computational Intelligence*, vol. 625, 2016, ch. RotorS—A, pp. 595–625, ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23.

[11] M. S. Kamel, T. Stastny, K. Alexis, and R. Siegwart, "Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System," in *Studies in Computational Intelligence*, 2017. DOI: 10.1007/978-3-319-54927-9_1.

[12] G. Foderaro, S. Ferrari, and T. A. Wettergren, "Distributed optimal control for multi-agent trajectory optimization," *Automatica*, vol. 50, no. 1, pp. 149–154, 2014. DOI: https://doi.org/10.1016/j.automatica.2013.09.014.

[13] R. Van Parys and G. Pipeleers, "Distributed MPC for multi-vehicle systems moving in formation," *Robotics and Autonomous Systems*, vol. 97, 2017. DOI: 10.1016/j.robot.2017.08.009.

[14] F. Rahimi and R. Esfanjani, "Distributed predictive control for formation of networked mobile robots," in *RSI International Conference on Robotics and Mechatronics (IcRoM*, Tehran, 2019. DOI: 10.1109/ICRoM.2018.8657625.

[15] J. Kim and Y. Kim, "Moving ground target tracking in dense obstacle areas using UAVs," *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 17, no. 1 PART 1, pp. 1–9, 2008. DOI: 10.3182/20080706-5-KR-1001.0214.

[16] P. Yao, H. Wang, and Z. Su, "Cooperative path planning with applications to target tracking and obstacle avoidance for multi-UAVs," *Aerospace Science and Technology*, vol. 54, pp. 10–22, 2016. DOI: 10.1016/j.ast.2016.04.002.

[17] Skydio, *Introducing the Skydio Autonomy Engine - YouTube*. [Online]. Available: https://www.youtube.com/watch?v=Gh5pAT1o2V8 (visited on 08/24/2020).

[18] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[19] X. Fu, H. Bi, and X. Gao, "Multi-UAVs Cooperative Localization Algorithms with Communication Constraints," *Mathematical Problems in Engineering*, vol. 2017, 2017. DOI: 10.1155/2017/1943539.

[20] J. C. Trujillo, R. Munguía, E. Ruiz-Velázquez, and B. Castillo-Toledo, "A Cooperative Aerial Robotic Approach for Tracking and Estimating the 3D Position of a Moving Object by Using Pseudo-Stereo Vision," *Journal of Intelligent and Robotic Systems: Theory and Applications*, pp. 297–313, 2019. DOI: 10.1007/s10846-019-00981-8.

[21] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-Novem, pp. 5332–5339, 2016. DOI: 10.1109/IROS.2016.7759784.

[22] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," Pittsburgh, PA, USA: International Conference on Embedded Software (EMSOFT), 2016, pp. 1–10. DOI: 10.1145/2968478.2968502.

[23] V. Nayak and R. R. Karaya, "Target Tracking by a Quadrotor Using Proximity Sensor Fusion Based on a Sigmoid Function," *IFAC-PapersOnLine*, vol. 51, no. 1, pp. 154–159, 2018. DOI: 10.1016/j.ifacol.2018.05.026.

[24] M. Rabah, A. Rohan, S. A. Mohamed, and S. H. Kim, "Autonomous Moving Target-Tracking for a UAV Quadcopter Based on Fuzzy-PI," *IEEE Access*, vol. 7, pp. 38 407–38 419, 2019, ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2906345.

[25] A. C. Woods and H. M. La, "Dynamic Target Tracking and Obstacle Avoidance using a Drone," vol. 2, pp. 857–866, 2015. DOI: 10.1007/978-3-319-27857-5.

[26] J. Chen and S. Shen, "Using a quadrotor to track a moving target with arbitrary relative motion patterns," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 5310–5317, 2017. DOI: 10.1109/IROS.2017.8206424.

[27] J. Chen, T. Liu, and S. Shen, "Tracking a Moving Target in Cluttered Environments Using a Quadrotor," *IEEE International Conference on Intelligent Robots and Systems*, vol. November, pp. 446–453, 2016. DOI: 10.1109/IROS.2016.7759092.

[28] X. Fu, K. Liu, and X. Gao, "Multi-UAVs Communication-Aware Cooperative Target Tracking," *Applied Sciences*, vol. 8, no. 6, 2018. DOI: 10.3390/app8060870.

[29] Z. He and J. Xu, "Moving target tracking by UAVs in an urban area," in *2013 10th IEEE International Conference on Control and Automation (ICCA)*, 2013, pp. 1933–1938. DOI: 10.1109/ICCA.2013.6564973.

[30] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian Optimization for Motion Planning," *Int. J. Rob. Res.*, vol. 32, no. 9-10, pp. 1164–1193, 2013. DOI: 10.1177/0278364913488805.

[31] V. Usenko, L. Von Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 215–222, 2017. DOI: 10.1109/IROS.2017.8202160. arXiv: 1703.01416.

[32] F. Gao, Y. Lin, and S. Shen, "Gradient-based online safe trajectory generation for quadrotor flight in complex environments," *IEEE International Conference on Intelligent Robots and Systems*, vol. September, pp. 3681–3688, 2017. DOI: 10.1109/IROS.2017.8206214.

[33] Y. Dong, C. Fu, and E. Kayacan, "RRT-based 3D path planning for formation landing of quadrotor UAVs," *2016 14th International Conference on Control, Automation, Robotics and Vision, ICARCV 2016*, vol. 2016, no. November, pp. 1–6, 2016. DOI: 10.1109/ICARCV.2016.7838567.

[34] Z. Hou, W. Wang, G. Zhang, and C. Han, "A survey on the formation control of multiple quadrotors," *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence, URAI 2017*, pp. 219–225, 2017. DOI: 10.1109/URAI.2017.7992717.

[35] A. Mahmood and Y. Kim, "Leader-following formation control of quadcopters with heading synchronization," *Aerospace Science and Technology*, vol. 47, pp. 68–74, 2015. DOI: 10.1016/j.ast.2015.09.009.

[36] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[37] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. DOI: 10.1007/BF01386390.

[38] R. Bellman, *Dynamic Programming*, ser. Rand Corporation research study. Princeton University Press, 1957, ISBN: 9780691079516.

[39] L. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, pp. 566–580, 1996. DOI: 10.1109/70.508439.

[40] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, IEEE Publishing, 1985, pp. 500–505. DOI: 10.1109/ROBOT.1985.1087247.

[41] C. W. Warren, "Global path planning using artificial potential fields," in *Proceedings, 1989 International Conference on Robotics and Automation*, 1989, 316–321 vol.1. DOI: 10.1109/ROBOT.1989.100007.

[42] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *Proceedings IEEE International Conference on Robotics and Automation*, 1993, pp. 802–807. DOI: 10.1109/robot.1993.291936.

[43] E. Rimon and D. E. Koditschek, "Exact Robot Navigation using Artificial Potential Functions," *IEEE Transactions on Robotics and Automation*, 1992. DOI: 10.1109/70.163777.

[44] M. P. Kelly, "Transcription Methods for Trajectory Optimization: a beginners tutorial," 2017, [Online]. Available: http://arxiv.org/abs/1707.00284.

[45] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, Second. Philadelphia: Society for Industrial and Applied Mathematics (SIAM), 2010, ISBN: 9780898716887.

[46] L. Petrović, "Motion planning in high-dimensional spaces," *eprint arXiv:1806.07457*, 2018. arXiv: 1806.07457. [Online]. Available: http://arxiv.org/abs/1806.07457.

[47] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," vol. 2015, pp. 42–49, 2015. DOI: 10.1109/ICRA.2015.7138978.

[48] M. Cap, P. Novak, J. Vokrinek, and M. Pechouvek, "Multi-agent RRT: Sampling-based Cooperative Pathfinding," *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1–2, 2013. arXiv: arXiv:1302.2828v1.

[49] D. Silver, "Cooperative Pathfinding.," in *Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2005*, 2005, pp. 117–122.

[50] T. Standley and R. E. Korf, "Complete Algorithms for Cooperative Pathfinding Problems.," in *IJCAI International Joint Conference on Artificial Intelligence*, 2011, pp. 668–673. DOI: 10.5591/978-1-57735-516-8/IJCAI11-118.

[51] H. Lee and H. J. Kim, "Trajectory tracking control of multirotors from modelling to experiments: A survey," *International Journal of Control, Automation and Systems*, vol. 15, no. 1, pp. 281–292, 2017. DOI: 10.1007/s12555-015-0289-3.

[52] B. Rubí, R. Pérez, and B. Morcego, "A Survey of Path Following Control Strategies for UAVs Focused on Quadrotors," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 98, no. 2, pp. 241–265, 2020, ISSN: 15730409. DOI: 10.1007/s10846-019-01085-z.

[53] M. Neunert, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, no. ICRA, pp. 1398–1404, 2016. DOI: 10.1109/ICRA.2016.7487274.

[54] A. Toloei and S. Niazi, "State Estimation for Target Tracking Problems with Non-linear Kalman Filter Algorithms," *International Journal of Computer Applications*, vol. 98, no. 17, pp. 30–36, 2014. DOI: 10.5120/17277-7708.

[55] Allauthor.com, *Peter Ducker Quotes*. [Online]. Available: https://allauthor.com/quote/107875/ (visited on 10/06/2020).

[56] A. Chatterjee, A. Rakshit, and N. N. Singh, "Simultaneous Localization and Mapping (SLAM) in Mobile Robots," in *Vision Based Autonomous Robot Navigation: Algorithms and Implementations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 167–206. DOI: 10.1007/978-3-642-33965-3_7.

[57] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, 2013. DOI: 10.1007/s10514-012-9321-0.

[58] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, no. November, pp. 1366–1373, 2017. DOI: 10.1109/IROS.2017.8202315.

[59] J. T. Betts, *A Survey of Numerical Methods for Trajectory Optimization*, 1998.

[60] M. Kelly, "Introduction to Trajectory Optimization," pp. 1–44, DOI: 10.1137/16M1062569.

[61] S. Flöry, "Fitting curves and surfaces to point clouds in the presence of obstacles," *Computer Aided Geometric Design*, vol. 26, no. 2, pp. 192–202, 2009. DOI: https://doi.org/10.1016/j.cagd.2008.04.003.

[62] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1476–1483, 2016. DOI: 10.1109/ICRA.2016.7487283.

[63] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental Trajectory Optimization for Real-time Replanning in Dynamic Environments," in *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 2012.

[64] L. Han, F. Gao, B. Zhou, and S. Shen, "FIESTA: Fast Incremental Euclidean Distance Fields for Online Motion Planning of Aerial Robots," in *Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 4423–4430. DOI: 10.1109/iros40897.2019.8968199. arXiv: 1903.02144.

[65] D. Mellinger and V. Kumar, "Minimum Snap Trajectory Generation and Control for Quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, Bahcesehir University, Grid Telekom, HUAWEI Techno, ISBN: 9781457718991.

[66] J. Ferrin, R. Leishman, R. Beard, and T. McLain, "Differential flatness based control of a rotorcraft for aggressive maneuvers," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2688–2693, 2011. DOI: 10.1109/IROS.2011.6048861.

[67] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006. DOI: 10.1007/s10107-004-0559-y.

[68] R. Tedrake, *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832)*, 2020. [Online]. Available: http://underactuated.mit.edu/ (visited on 08/20/2020).

[69] M. Behrendt, *MPC scheme basic*, Oct. 2009. [Online]. Available: https://commons.wikimedia.org/wiki/File:MPC%7B%5C_%7Dscheme%7B%5C_%7Dbasic.svg (visited on 09/18/2020).

[70] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019. DOI: 10.1007/s12532-018-0139-4.

[71] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter, "An algorithmic framework for convex mixed integer nonlinear programs," *Discrete Optimization*, vol. 5, no. 2, pp. 186–204, 2008. DOI: 10.1016/j.disopt.2006.10.011.

[72] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Knitro: An Integrated Package for Nonlinear Optimization," in, Springer, Boston, MA, 2006, pp. 35–59. DOI: 10.1007/0-387-30065-1_4.

[73] C. Büskens and D. Wassel, "The ESA NLP solver WORHP," in *Springer Optimization and Its Applications*, vol. 73, Springer International Publishing, 2013, pp. 85–110. DOI: 10.1007/978-1-4614-4469-5_4.

[74] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.

[75] *Find minimum of constrained nonlinear multivariable function - MATLAB fmincon*. [Online]. Available: https://www.mathworks.com/help/optim/ug/fmincon.html (visited on 02/27/2020).

[76] Nexcis, *Lagrange Multipliers2D - Wikimedia Commons*, Dec. 2017. [Online]. Available: https://commons.wikimedia.org/wiki/File:LagrangeMultipliers2D.svg (visited on 09/17/2020).

[77] DJI, *DJI - Camera Drones/Quadcopters for Aerial Photography*. [Online]. Available: https://www.dji.com/ (visited on 08/26/2020).

[78] The SciPy community, *numpy.arctan2 — NumPy v1.19 Manual*, Jun. 2020. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.arctan2.html (visited on 10/02/2020).

[79] Productz, *AscTec Firefly*. [Online]. Available: `https://productz.com/en/asctec-firefly/p/w9DR` (visited on 07/24/2020).

[80] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: a parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, Dec. 2014. DOI: `10.1007/s12532-014-0071-1`.

[81] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, vol. 3, 2009.

[82] C. Taylor, *What Is the Interquartile Range Rule?* 2018. [Online]. Available: `https://www.thoughtco.com/what-is-the-interquartile-range-rule-3126244` (visited on 08/30/2020).

[83] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos, "A simple and efficient algorithm for nonlinear model predictive control," *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, vol. January, 2018. DOI: `10.1109/CDC.2017.8263933`.

[84] A. Wächter, "Short Tutorial: Getting Started With Ipopt in 90 Minutes," in *Combinatorial Scientific Computing*, ser. Dagstuhl Seminar Proceedings, Dagstuhl, Germany: Dagstuhl Seminar Proceedings 09061, 2009.

Appendix A

Constraint Optimisation

Section 3.9 provided an introduction on how an unconstrained NLP is solved using Newton's method. This appendix expands on the section by introducing how constraints are enforced on the optimisation problem. The specific form of the optimisation problem IPOPT solver is given in Figure A.1. The information in this appendix is adapted from Betts [45] and Wächter and Biegler [67].

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{R}^n} \quad & F(\boldsymbol{x}) \\ \text{s.t.} \quad & g^L \leq \boldsymbol{g}(\boldsymbol{x}) \leq g^U \\ & \boldsymbol{h}(\boldsymbol{x}) = 0 \\ & x^L \leq \boldsymbol{x} \leq x^U, \end{aligned} \tag{A.1}$$

A.1 Constraints

This section explains how the *Lagrangian method* can be used to enforce equality constraints on the optimisation problem, and how inequality constraints can be represented as equality constraints.

A.1.1 Equality constraints

A popular way to enforce equality constraints in solving an NLP is to make use of the *Lagrangian method*. Specifically, to solve optimisation problems in the form

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{R}^n} \quad & F(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}) = 0 \end{aligned} \tag{A.2}$$

where $\boldsymbol{h}$ is the constraint function that needs to be satisfied. To solve the optimisation problem, a new function $\mathcal{L}$, called the Lagrangian is introduced, defined as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = F(\boldsymbol{x}) - \boldsymbol{\lambda}^\intercal \boldsymbol{h}(\boldsymbol{x}) \tag{A.3}$$

where $\boldsymbol{\lambda}$ is a vector called the Lagrange multiplier. The core idea behind the Lagrange multiplier technique is to find the points where the contour lines of the objective function F and the constraint function $\boldsymbol{h}$ are tangent to each other [45]. To calculate this point, the gradient of $\mathcal{L}$ is set equal to the zero vector, therefore

$$\nabla\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \boldsymbol{0} \tag{A.4}$$

It is also important to check that the point is indeed the minimum, and not a maximum, which can be checked by

$$\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}, \tag{A.5}$$
$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}. \tag{A.6}$$

Equations A.4, A.5, and A.6 together form a set of matrix equations, which, when solved with a numerical algorithm, give the solution to the constrained optimisation problem.

A.1.2 Inequality constraints

The second type of constraints is inequality constraints, specifically in the form

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{R}^n} \quad & F(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}^L \leq \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{g}^U \\ & \boldsymbol{h}(\boldsymbol{x}) = 0 \\ & \boldsymbol{x}^L \leq \boldsymbol{x} \leq \boldsymbol{x}^U, \end{aligned} \tag{A.7}$$

IPOPT handles inequality constraints by internally converting them to equality constraints with a bounded slack variable, meaning that the constraint $\boldsymbol{g}^L \leq \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{g}^U$ is converted to $g_i(\boldsymbol{x}) - s_i = 0$, where s_i is the newly introduced slack variable. The variable s_i is bounded between $\boldsymbol{g}_i^L \leq s_i \leq \boldsymbol{g}_i^U$. Equation A.1 can be converted to

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{R}^n} \quad & F(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{c}(\boldsymbol{x}) = 0 \\ & \boldsymbol{x}^L \leq \boldsymbol{x} \leq \boldsymbol{x}^U, \end{aligned} \tag{A.8}$$

where $\boldsymbol{x}$ now includes the slack variables s, and $\boldsymbol{c}(\boldsymbol{x})$ is a set of equality constraints (containing $g_i(\boldsymbol{x}) - s_i = 0$). The upper and lower bounds $x^L \leq \boldsymbol{x} \leq \boldsymbol{x}^U$ now include the bounds of the slack variable, $\boldsymbol{g}_i^L \leq s_i \leq \boldsymbol{g}_i^U$.

A.2 Interior-point method

Instead of immediately solving the complete NLP with constraints, the optimisation algorithm starts by solving a relaxed version of the original problem. With this approach, the convergence of the optimisation algorithm is better if the problem is ill-posed. This is achieved with the introduction of a barrier function, which is multiplied by a barrier parameter μ. Boundary constraints are when the decision variables are bounded to an upper bound and lower bound in the form

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathcal{R}^n} \quad & F(\boldsymbol{x}) \\ \text{s.t.} \quad & x^L \leq \boldsymbol{x} \leq x^U. \end{aligned} \tag{A.9}$$

With an interior point method, a barrier function is introduced to enforce the bounds on the decision variable x. The formulation of the NLP problem can be further rewritten as

$$\min_{\boldsymbol{x}\in\mathcal{R}^n} \quad \varphi_\mu(\boldsymbol{x}) = F(\boldsymbol{x}) - \mu \sum_{i=1}^{n} \ln(b_i - x_i) \tag{A.10}$$

where a logarithmic barrier term replaces the boundary constraints. The barrier function $(-\mu \sum_{i=1}^{n} \ln(b_i - x_i))$ is designed to go to infinity if any of the variables x_i approaches their bound b_i.

Starting with a moderate value of μ (e.g., 0.1) and a user-supplied starting point, the corresponding barrier problem in Equation A.10 is solved to relaxed accuracy [67]. The barrier parameter μ is then decreased and solved with a tighter accuracy. This process is repeated until the solution for Equation A.8 is reached, with $\mu = 0$.

The algorithm starts with an initial value ($\mu \approx 0.1$), which is slowly lowered at various iterations. When $\mu = 0$, it does not have an impact on the optimisation problem anymore, yet it ensures that the algorithm converges to the desired constraints. Figure A.1 shows the barrier parameter for the example presented in Section 4.10. As expected, the μ is slowly decreased until the solution is reached.

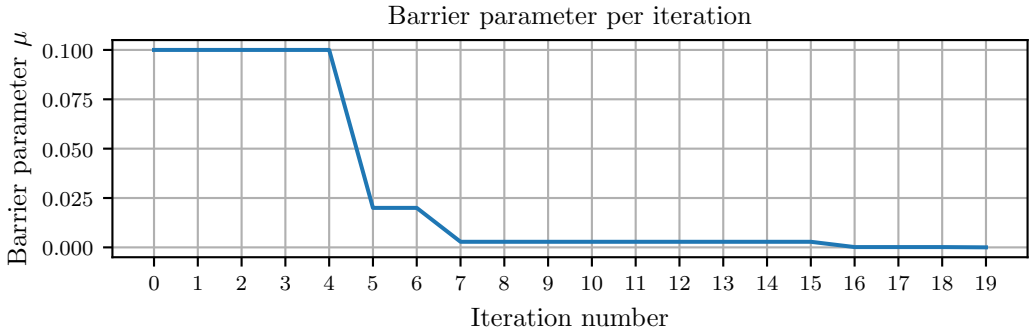


Figure A.1: Barrier parameter for each iteration.

A.3 Globalisation strategies

The technique above offers fast convergence if the problem is well posed, however, the performance degrades if the functions are not strictly convex [45]. In optimisation, there are a few strategies (which IPOPT also employs), to correct for some of the deficiencies in the standard formulation. These are called globalisation strategies.

One of the strategies IPOPT employs is called a *line-search strategy*. In Newton's method, at each iteration the gradient information tells the optimiser in which direction to take a step, and how big the step should be. The line-search strategy works instead by giving a single steps; it first tries a few step sizes, and then selects the step that makes the most progress.

A.3.1 Merit function

If Newton's method is working correctly, the sequence of iterations $\boldsymbol{x}(k)$ should converge to the solution $\boldsymbol{x}*$. In practice, of course, the solution $\boldsymbol{x}*$ is unknown, and we must detect whether the sequence is converging using available information [45]. A common way to measure progress is to assign some *merit function*, M, and insist that it decreases with each iteration, symbolically meaning

$$M(\boldsymbol{x}^{(k+1)}) < M(\boldsymbol{x}^{(k)}). \tag{A.11}$$

When using Newton's method for unconstrained optimisation, an obvious choice for the merit function is just to use the objective function, so that $M(\boldsymbol{x}) = f(\boldsymbol{x})$. However, choosing a merit function for constrained optimisation is more problematic. It is often necessary to balance conflicting goals of reducing the objective function while satisfying the constraints.

A.3.2 Line search

Assuming that the merit function can measure progress, how can the search be altered if progress is not being made? One way is to alter the magnitude of the step using a *line-search* method. The basic Newton step is given in an iterative form of

$$\bar{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{p}, \tag{A.12}$$

where $\mathbf{p}$ is defined as the search direction. The line-search technique works by replacing the Newton step with

$$\bar{\boldsymbol{x}} = \boldsymbol{x} + \alpha\boldsymbol{p}, \tag{A.13}$$

By adjusting α, the magnitude of the step is adjusted. The line-search method works by trying different step sizes and then selects the step that results in the most progress.

A.3.3 Filters

But how does the optimiser determine the step size that makes the most progress? In constrained optimisation, there are two conflicting aims. The first is to minimise the objective function, and the second is to minimise the constraint violation [45]. In some cases, the optimisation strategy has to sacrifice the objective function to satisfy the constraints. Similarly, by momentarily increasing the constraint violation, it may make significant reductions to the objective function. IPOPT makes use a *filter strategy* for deciding if the Newton iterate is working properly [84].

The basic idea is to compare the information of the current iteration to the information of the previous iteration and then "filter" out the bad iterates. The algorithm keeps a list of the previous iterations, both in terms of the objective function $F(\boldsymbol{x})$ and the constraint violation $v[\mathbf{c}(\boldsymbol{x})]$. With the filter technique, a Newton step will be accepted if it decreases either the objective function or the constraint violation.

To illustrate this concept, an example is given in Figure A.2. On the vertical axis the objective function, $F(\boldsymbol{x})$, is indicated, and the horizontal axis shows the constraint violation, $v[\mathbf{c}(\boldsymbol{x})]$. The first filter point is marked as "Filter Point 1". Now, any new point that falls within the grey region behind that point will not be accepted, due to it being worse than the current point. If another point is found, "Filter Point 2", it is added to the filter list. Now any new point should be better than both filter points to be considered as making satisfactory progress. This process is repeated until a point is found which lies sufficiently close to the solution point, and the $v[\mathbf{c}(\boldsymbol{x})] = 0$.

In some cases, it is possible that no step size can be found which falls within the line-filter point. IPOPT handles this by switching to a *feasibility restoration phase*. The objective function gets ignored temporarily, and instead solves a different optimisation problem, minimising the constraint violation $v[\mathbf{c}(\boldsymbol{x})]$ [84]. After the feasibility restoration phase, IPOPT can return to solving the original optimisation problem. However, if the constraints cannot be satisfied, it will indicate to the user that the problem is (locally) infeasible.

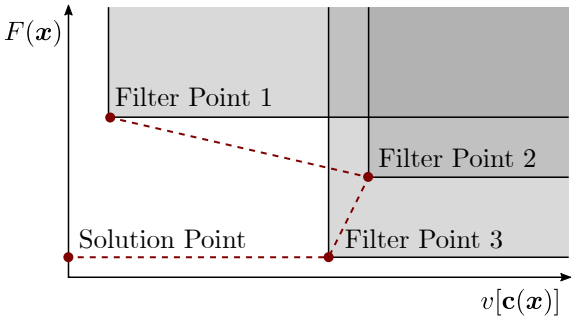


Figure A.2: Line search filter strategy, adapted from Betts [45].

To illustrate the line-search filter, Figure A.3 plots the *objective function* against the *constraint violation* at each iteration, for the optimal control problem presented in Section 4.10. The iterations slowly "snake" their way to the lower-left corner, where the constraints are satisfied, and the objective is the lowest. A zoomed-in plot is included to show that even when the changes are minimal, the same pattern is followed. The example verifies that the filter works as expected.

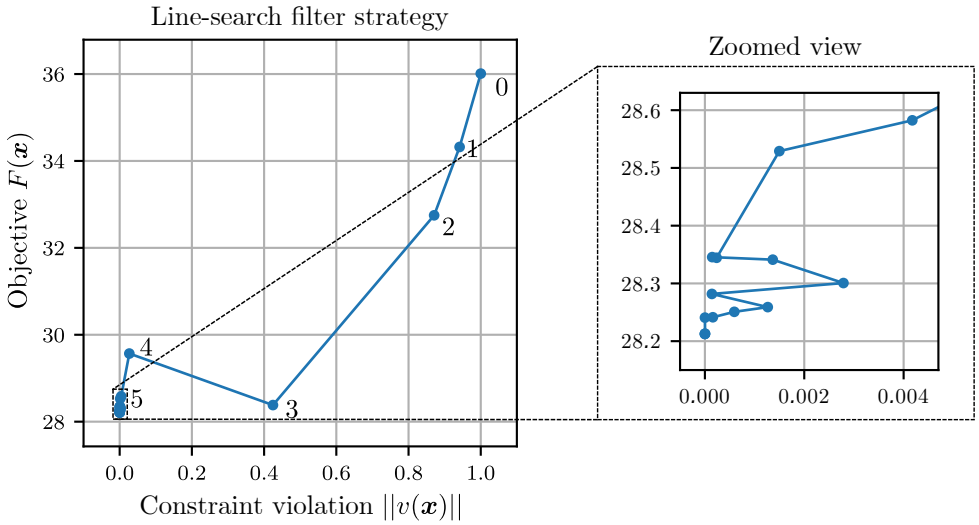


Figure A.3: Line-search filter for constraint optimisation.

Another interesting note is that the observations in Table 4.1 are confirmed in Figure A.3. For instance, the graph shows that initially much progress is made, and as the iterations continue the change per iteration is reduced. The jump between iteration three and four is also visible, in which the objective increased, but the constraint violation decreased significantly.

A.4 Gradient calculation

The ability to efficiently calculate the gradients of functions is fundamental to solving NLPs. A popular way to calculate the gradients required is to make use of Automatic Differentiation (AD) [70]. The advantage of AD is that it has the accuracy and speed of analytical methods for calculating derivatives, but the ease of use of numerical methods.

This project makes use of CasADi [70] to calculate the necessary gradient information for IPOPT. CasADi creates an expression graph of functions and then makes use of the

chain rule of differentiation to calculate the gradient. As an illustration, consider this example function $f(x_1, x_2) = x_1 x_2 + \sin(x_1)$. It is possible to create a corresponding expression graph as shown in Figure A.4.

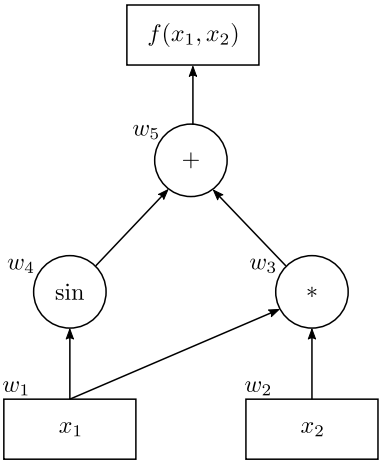


Figure A.4: Expression graph for performing automatic differentiation.

The expression graph can be used to calculate both the value of the function as well as the derivative. Table A.1 shows how this can be achieved. By breaking the expression down to simple expressions, it is possible to apply the chain rule to calculate the derivative of the function.

Table A.1: Expression graph for calculating derivative

Operations to compute value	Operations to compute derivative
$w_1 = x_1$	$\dot{w}_1 = 1$
$w_2 = x_2$	$\dot{w}_2 = 1$
$w_3 = w_1 \cdot w_2$	$\dot{w}_3 = \dot{w}_1 \cdot w_2 + w_1 \cdot \dot{w}_2$
$w_4 = \sin(w_1)$	$\dot{w}_4 = \cos(w_1) \cdot \dot{w}_1$
$w_5 = w_3 + w_4$	$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$

Appendix B

Rotary-wing UAV dynamics

A simulation model of the AscTec Firefly hexarotor is included in the RotorS simulator and is used in the next chapter to verify the performance of the trajectory planner. The information in this section has been adapted from Furrer *et al.* [10].

B.1 Mathematical model

The simulation model consists of a rigid body with six fixed rotors. Each rotor experiences forces and moments that act on it. These forces and moments are summarised in Figure B.1. The forces acting on the propeller are the thrust force $\boldsymbol{F_T}$ and the drag force $\boldsymbol{F_D}$. The moments acting on the rotor are the rolling moment $\boldsymbol{M_R}$ and the moment induced by drag $\boldsymbol{M_D}$. These values are functions of the angular velocity and aerodynamic properties of the rotors.

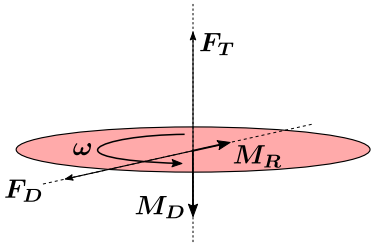


Figure B.1: Forces and moments acting on the center of a single rotor [10].

The complete UAV dynamics are described by the forces acting on the rotors and those acting on the UAV centre of mass. Through Newton's law and Euler's equation, the equations of motion can be expressed as

$$\begin{aligned} \boldsymbol{F} &= m \cdot \boldsymbol{a} \\ \boldsymbol{\tau} &= \boldsymbol{J} \cdot \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{J} \cdot \boldsymbol{\omega} \end{aligned} \tag{B.1}$$

where m is the mass of the UAV, $\boldsymbol{a}$ the acceleration, $\boldsymbol{J}$ its inertia matrix, and $\boldsymbol{\omega}$ the angular velocity. The linear part is expressed in the world coordinate system, which references to a predetermined fixed point in the environment. The rotational part is expressed in the rotating body frame, also fixed to the world coordinate system defined in the environment.

Figure B.2 illustrates a free body diagram for a multi-rotor helicopter. For a multi-rotor helicopter with n rotors, Equation B.1 can be rewritten as

$$\begin{aligned} &\sum_{i=0}^{n-1}(\boldsymbol{R_{WB}}\underbrace{(\boldsymbol{F_{T,i}}+\boldsymbol{F_{D,i}})}_{\boldsymbol{F_i}}+\boldsymbol{F_G}=m\cdot\boldsymbol{a} \\ &\sum_{i=0}^{n-1}(\boldsymbol{M_{R,i}}+\boldsymbol{M_{D,i}}+\boldsymbol{F_i}\times\boldsymbol{r_i})=\boldsymbol{J}\cdot\dot{\boldsymbol{\omega}}+\boldsymbol{\omega}\times\boldsymbol{J}\cdot\boldsymbol{\omega} \end{aligned} \tag{B.2}$$

where $\boldsymbol{R_{WB}}$ is the rotation matrix from the body frame B to the world frame W, and $\boldsymbol{r_i}$ denotes the vector from the centre of mass to the i-th rotor.

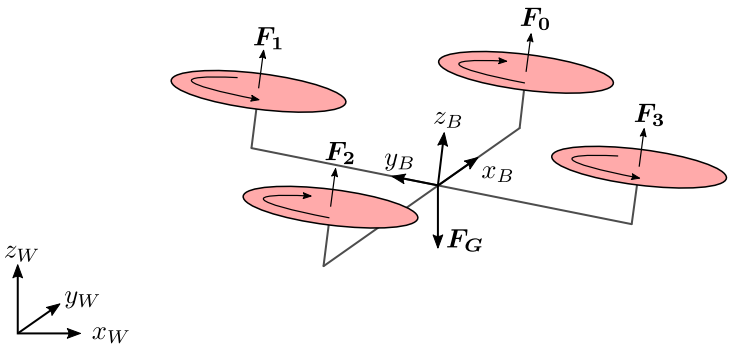


Figure B.2: Quadrotor with the body frame B and the global world frame W [10].